# SONiX 8-Bit MCU

# M2IDE 用户手册

Version 1.0

# 修改记录

| 版本 | 时间 | 修改说明 |
|---|---|---|
| V1.0 | 2009/8/18 | 初版 |

# 目　　录

# 前　　言

在 SONiX 8 位微控器中，SN8P2XXX 系列开发系统采用的是在线仿真器（ICE），而 PC 机上运行的软件是 M2IDE 或 SN8 C Studio。

本手册主要讲述 M2IDE 集成开发环境。

# 第 1 章 系统概要与安装

## 1.1 M2IDE简介

M2IDE 由编辑器、汇编器、调试器和芯片烧写等部分组成，可以完成工程建立和管理、代码编写、编译、链接、目标代码生成和硬件仿真。

用户可以登陆 www.sonix.com.tw ，下载最新版本的M2IDE安装文件。

**M2IDE 特点：**

- 程序指令的实时仿真
- 使用及安装方便
- 通用的视窗界面
- 支持多个源程序文件的工作平台(一个应用项目可以包含一个以上的源程序文件)
- 支持软件模拟 LCD 仿真

# 1.2 安装

## 1.2.1 系统配置需求

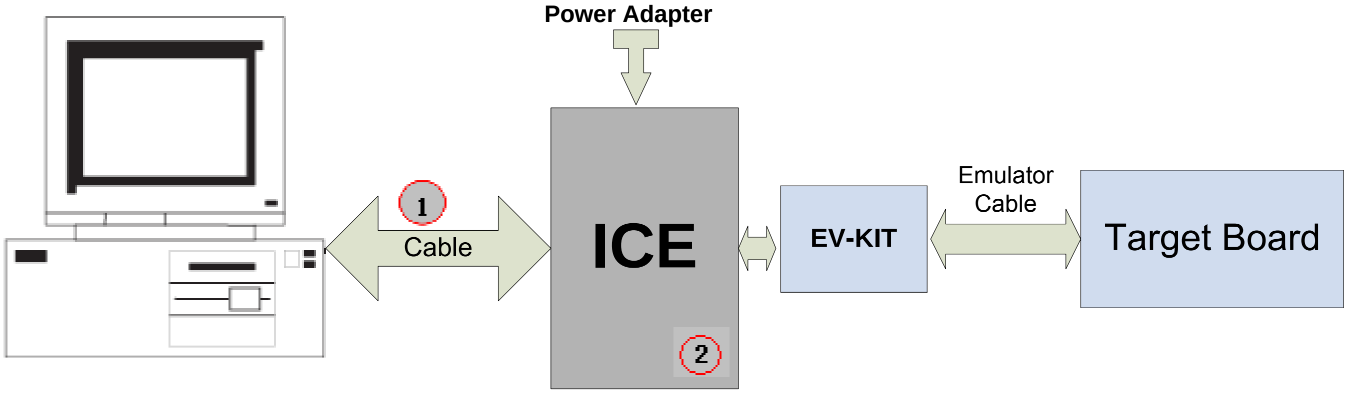



图 1-1 仿真连接示意图

图 1-1 中①、②处描述如下：

1. ①处 Cable 可以为 Printer Cable 或 USB Cable；
   SN8ICE 2K 提供 Printer 口与 PC 相连接；
   SN8ICE 2K Plus II 提供 USB 口与 PC 相连接；

   SONiX 还提供 UTP 装置，将 Printer Port 转换为 USB Port；

为确保 M2IDE 能够正常工作，安装环境需要满足最低配置要求，条件如下：

- Windows 98/2000/ XP/ Vista 操作系统
- 至少 32 MB 的可用硬盘空间
- 至少 32 MB 的内存空间

## 1.2.2 硬件安装

在进行硬件连接前请先确认晶振、振荡电路电容、短路帽等插接正确，详细信息请参考注意事项中的晶振安装相关内容。

**硬件连接前，请确认ICE的电源是关闭的！**

SN8ICE 2K 硬件如图 1-2：

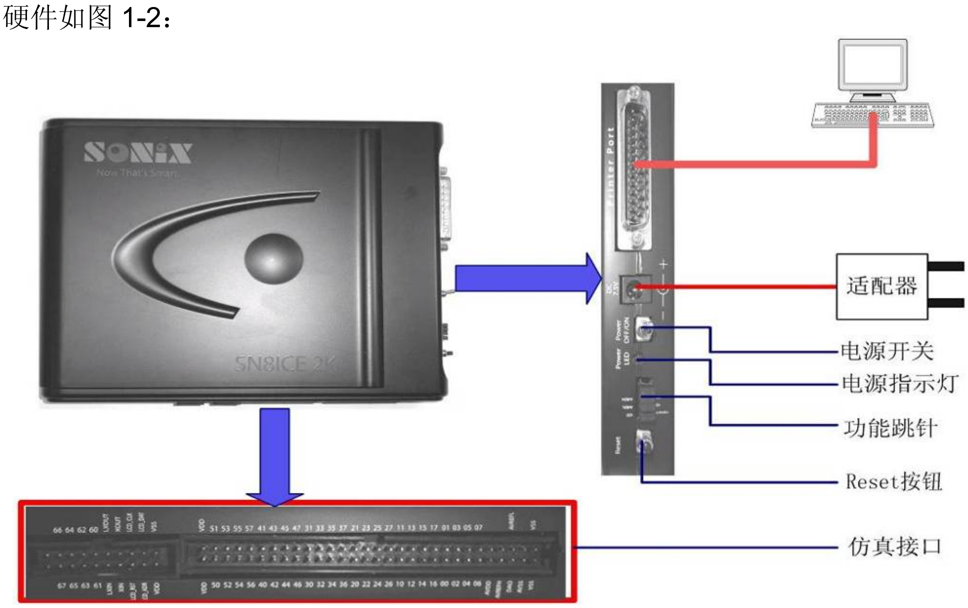



图 1-2 SN8ICE 2K 硬件说明

仿真器硬件安装按以下步骤进行：

1. 将 SN8ICE 2K / SN8ICE_USB / SN8ICE 2K Plus 在线仿真器和计算机 LPT 口通过并口线连接起来；
   笔记本电脑用户可以使用 SONiX 提供的 USB 转并口的工具（UTP）进行连接；
   SN8ICE 2K Plus II 在线仿真器通过 USB 线和计算机 USB 接口相连接；
2. 选择合适的 LPT 端口；
3. 仿真器加上 DC 电源（必须使用专用的直流电源）；
4. 安装开发软件 M2IDE_Vxxx.软件后便可编辑、编译或仿真程序；

连接后示意图如图 1-3 所示：

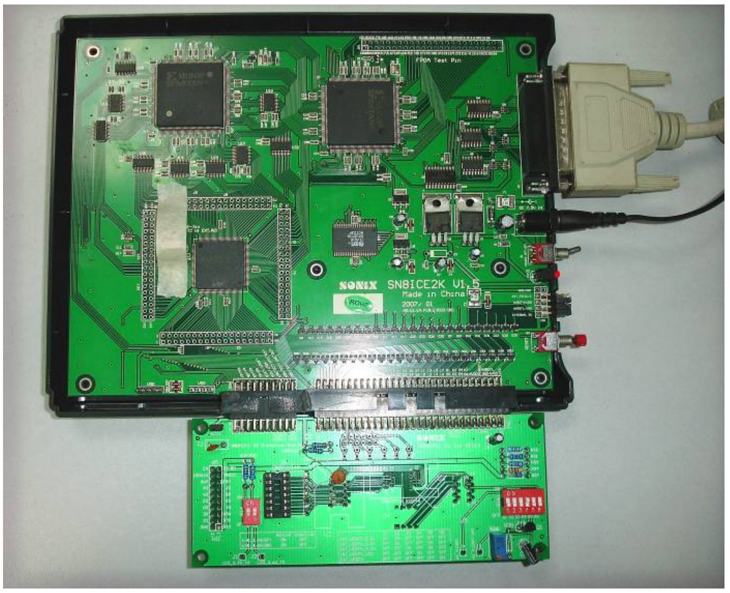


图 1-3 连接后示意图

有关仿真器的更多信息可以参考 SONiX's SN8ICE2K Easy GuideV1.0，用户可以在网上下载。

### 1.2.3 软件安装

安装文件名的格式如下：M2IDE_Vxxx.exe。其中M2IDE为软件包的名称，Vxxx为该软件的版本，例如M2IDE_V119。用户可以登陆 www.sonix.com.tw 下载最新版本的M2IDE安装文件。

下面以M2IDE_V119版本来说明M2IDE编译器的安装步骤：

双击安装文件M2IDE_V119.exe开始安装，这时会弹出图1-4所示的向导对话框，对话框中提示用户该开发环境适用的仿真器及芯片类型，并建议用户在安装的同时退出其它正在运行的程序，以保证安装的顺利进行。

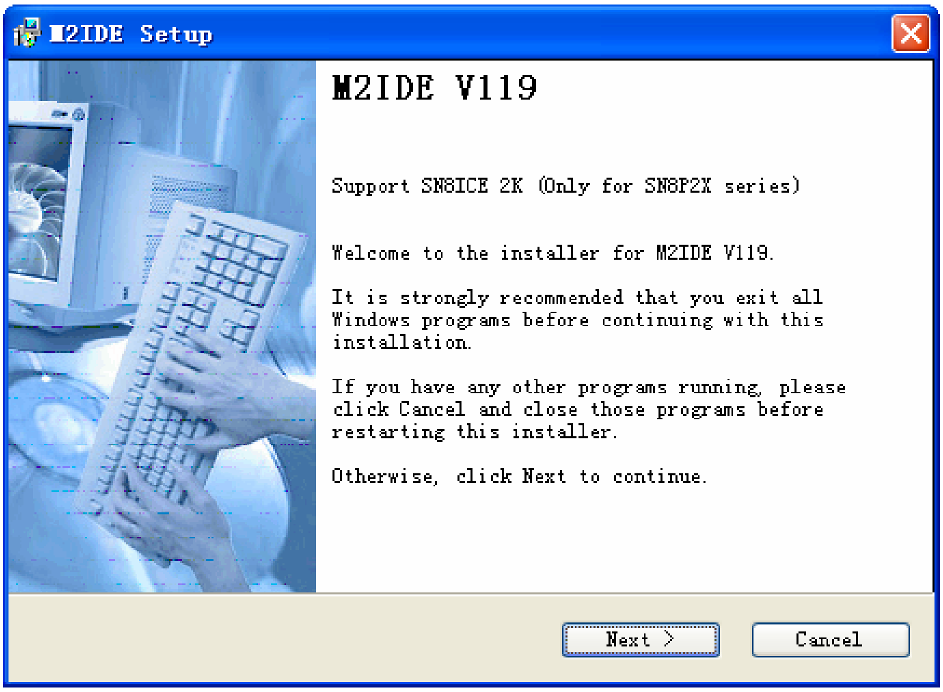



图 1-4 M2IDE 安装向导对话框

单击“Next”命令按钮，此时弹出如图1-5所示的协议对话框，要求用户仔细阅读软件使用的相关协议，要想继续安装该软件，必须选中“I agree to the terms of this license agreement”项。

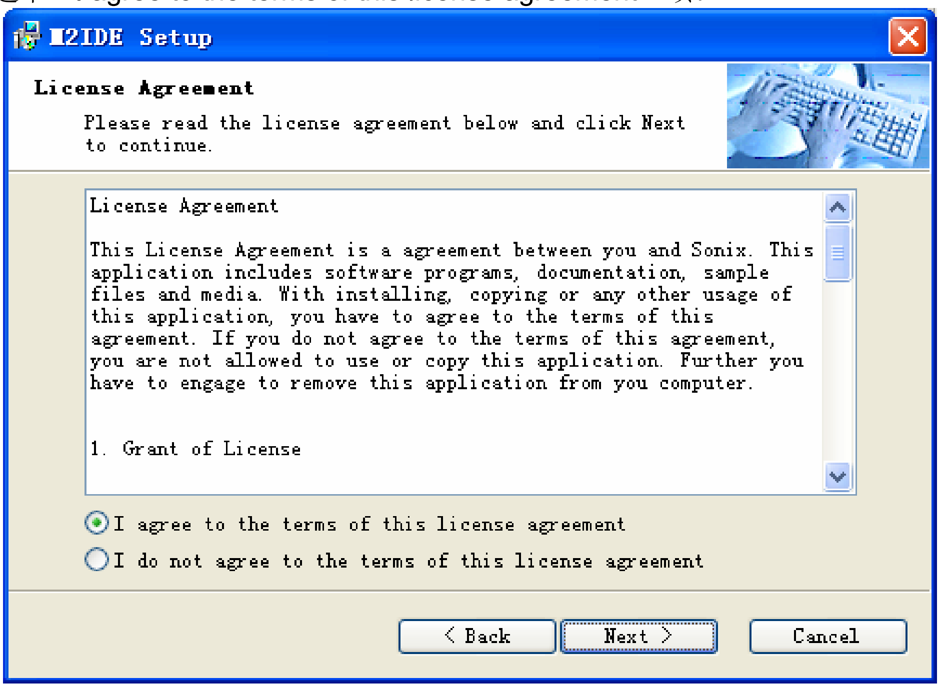



图 1-5 M2IDE 安装协议对话框

再次单击“Next”命令按钮，弹出如图1-6所示的安装路径选择对话框，默认的安装路径为C:\Sonix\M2IDE_V119，用户也可以通过单击“Change…”来改变安装路径，图1-7为单击“Change…”后弹出的路径浏览窗口，当指定相应的路径后单击“确定”完成路径的选择。

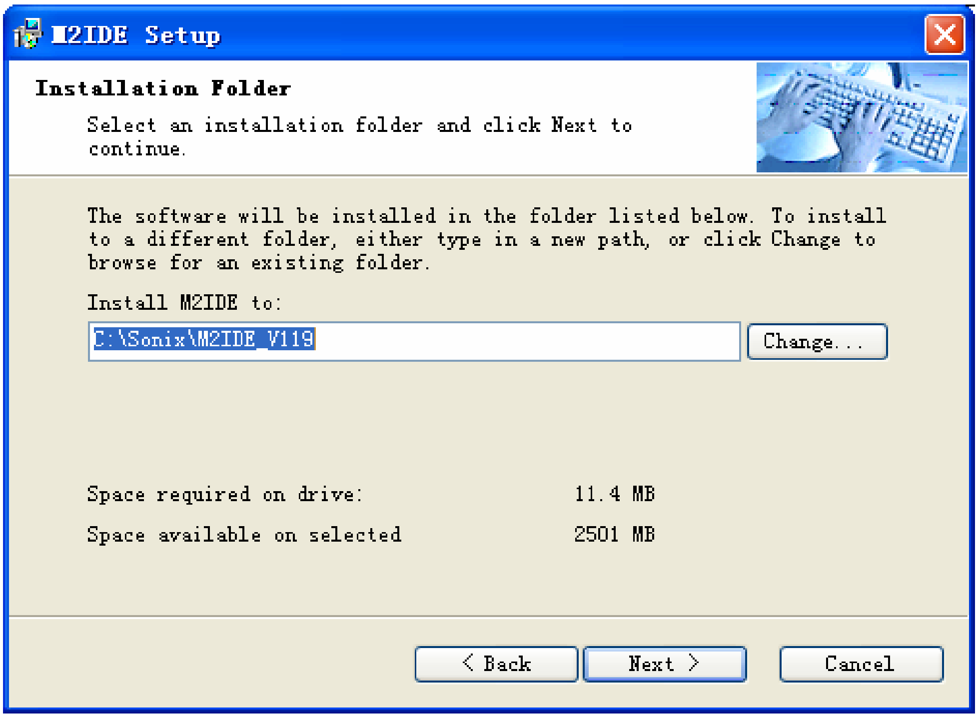



图 1-6 安装路径选择对话框




图 1-7 浏览路径窗口

继续单击“Next”命令按钮，弹出快捷方式所指向的文件夹设置窗口，如图1-8所示，安装文件会创建一个快捷方式，由此用户可以使其指向默认的文件夹，也可以使其指向一个新文件夹，或者直接在列表中选择。

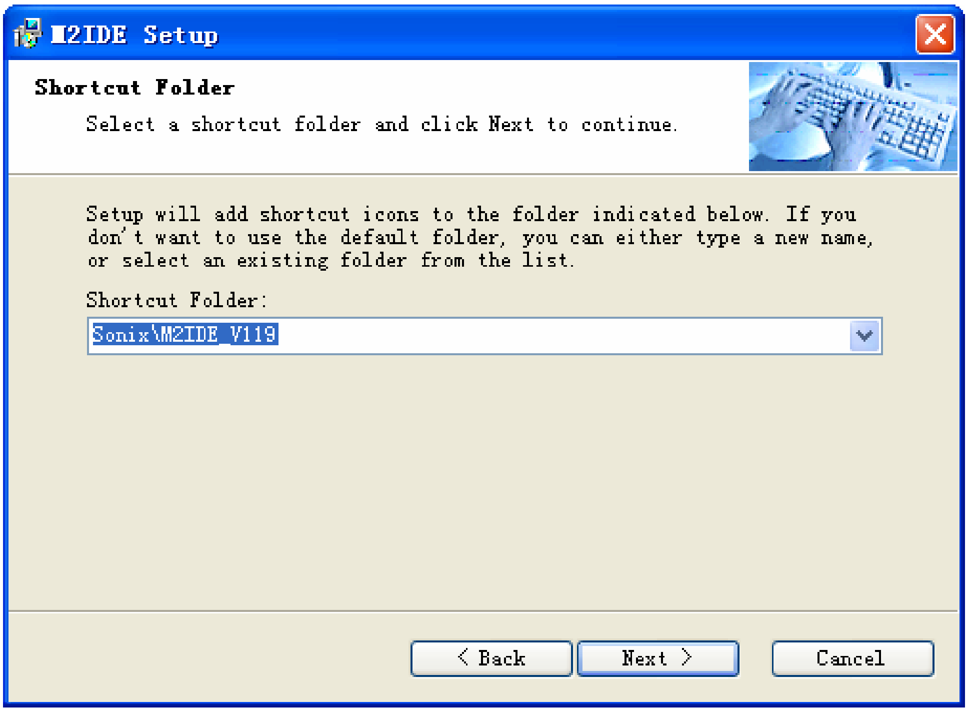



图 1-8 弹出快捷方式设置窗口

再次单击“Next”命令按钮，弹出安装配置信息对话框，如图1-9所示，其中包含了安装路径和快捷方式指向的相关信息，如果确认无误，则可以单击“Next”命令按钮进入下一步安装，此时在弹出安装进度窗口，如图1-10所示。

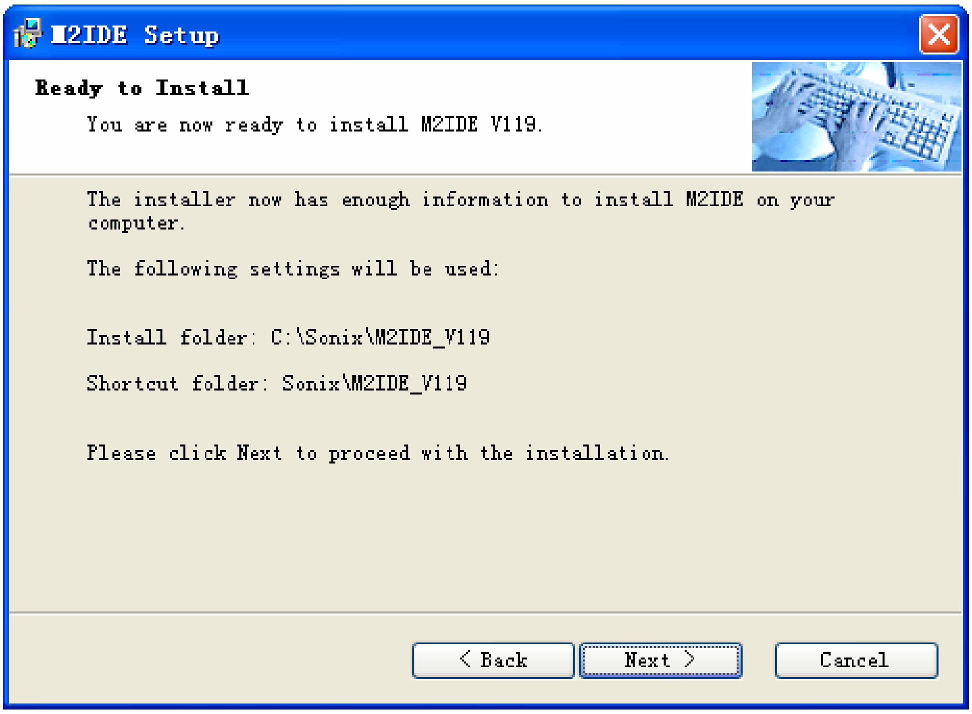



图 1-9 安装信息

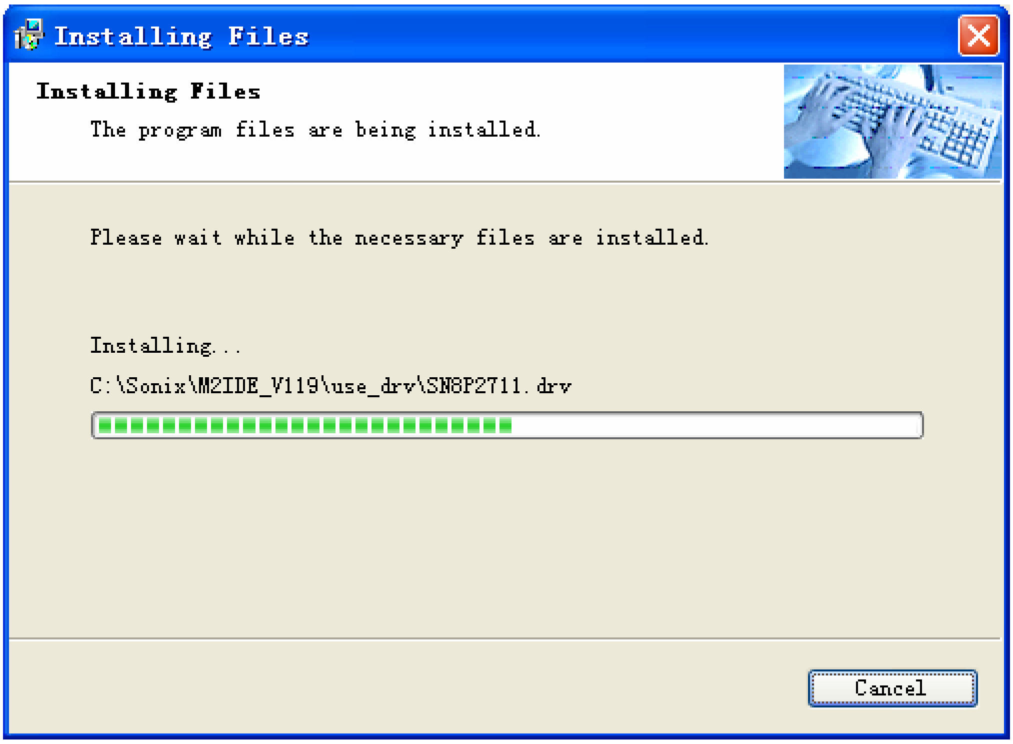



图 1-10 安装进度窗口

最后弹出安装结束窗口，如图1-11所示，表明程序已经成功安装，这时单击“Finish”命令按钮结束安装。此时在桌面上可以看到M2IDE_V119的快捷图标，如图1-12所示，通过双击它就可以进入集成开发环境。

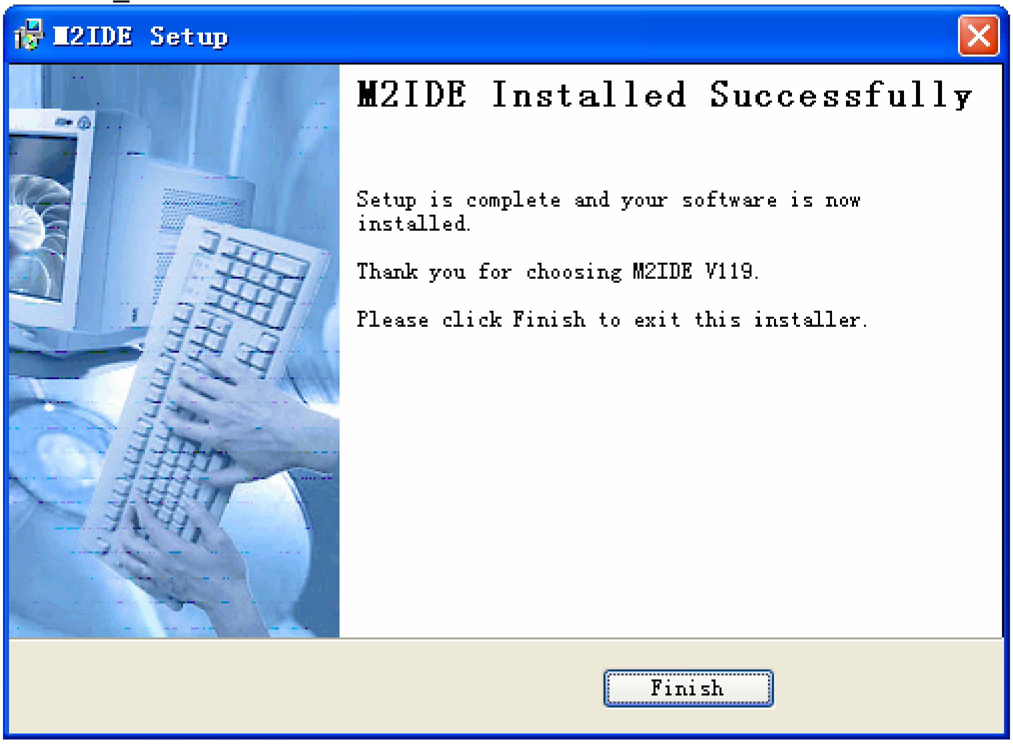



图 1-11 结束安装




图 1-12 快捷图标

安装完成 M2IDE 软件后，双击桌面上的快捷图标或单击：开始/程序/Sonix/M2IDE V119/M2Asm119.exe，即可进入集成开发环境。如图 1-13 所示。

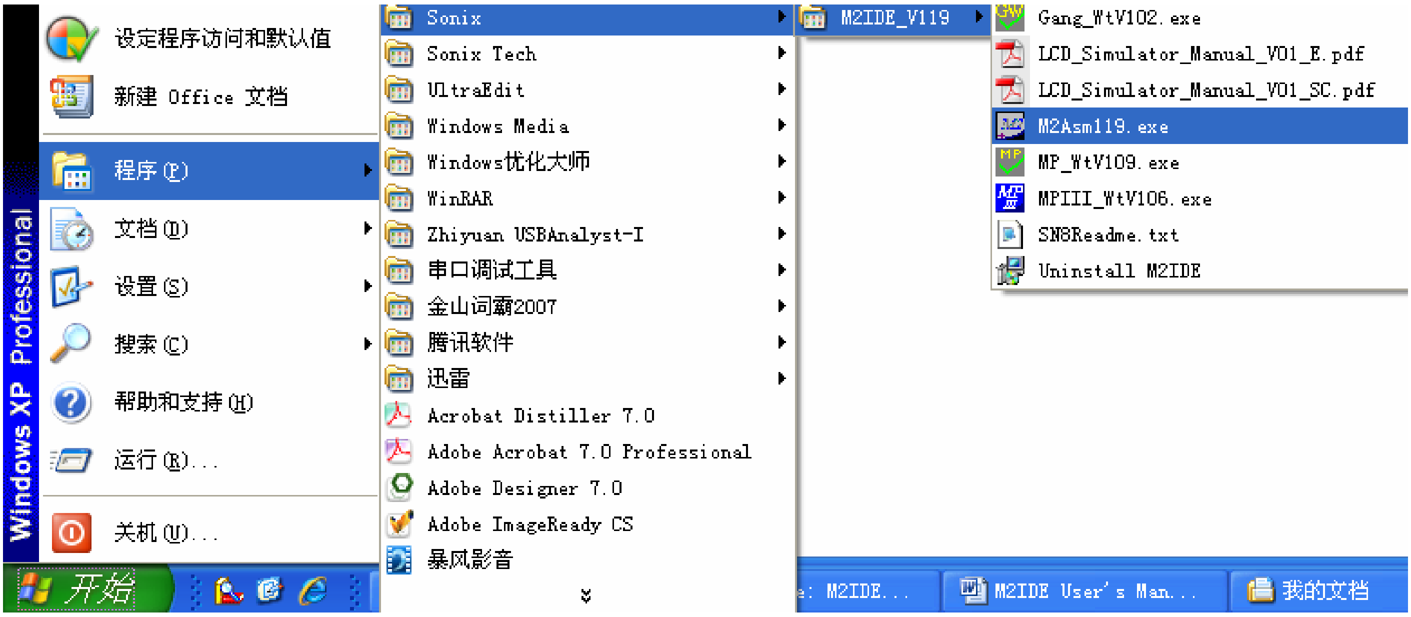



图 1-13 启动 M2IDE 系统

**M2IDE** 编译器卸载方法如下：
执行：开始/程序/Sonix/M2IDE V119/Uninstall M2IDE。

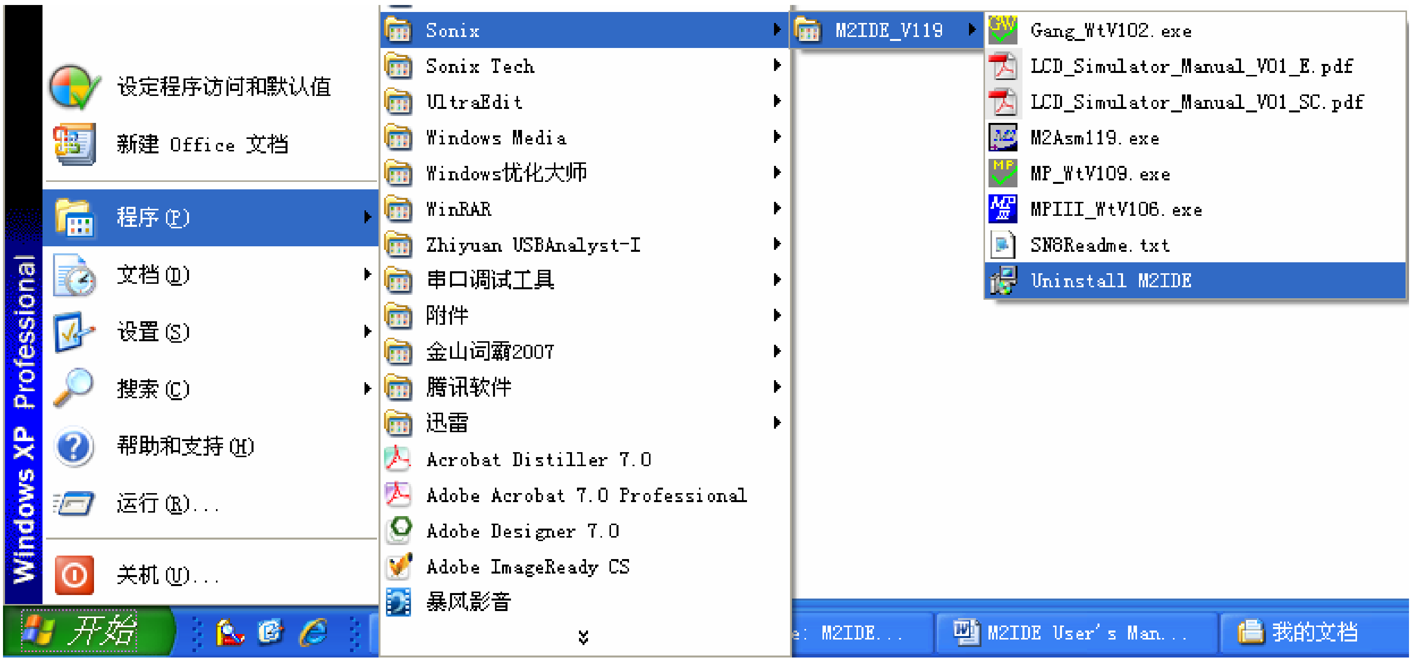



图 1-14 卸载 M2IDE

卸载成功后会弹出如下对话框。

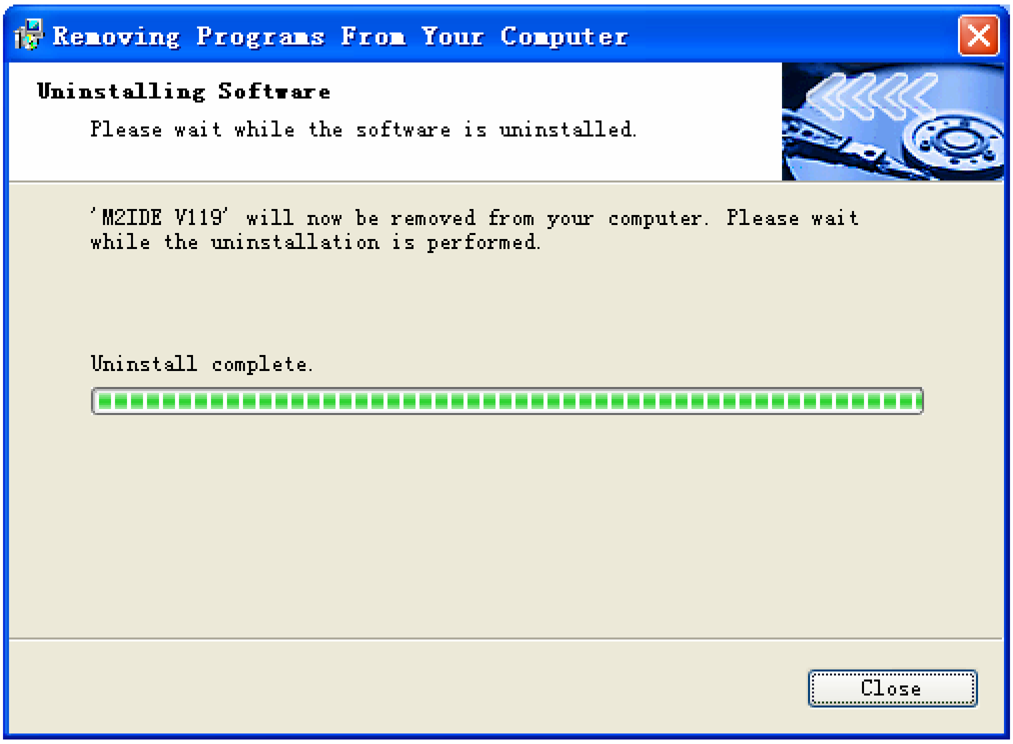



图 1-15 卸载成功提示信息框

# 第 2 章 视窗界面

## 2.1 快速开始

本节讲述如何快速去开发一个应用程序项目。
用并口线或 USB 线将仿真器和 PC 机相连接，并接上在线仿真器电源。

步骤一：新建工程

- [开始]→[程序]→[Sonix]→[M2IDE_V119]→M2Asm119.exe 打开M2IDE；
- [File]菜单→[New] 命令；
- [File]菜单→[Save] 命令或按下快捷键“Ctrl+S”；
- 选择需要保存的路径，输入项目名称，如Main.ASM；
- [File]菜单→[New Project]命令，打开已保存的源文件；

步骤二：源程序文件建立

- [File]菜单→[New] 命令，建立源程序文件；
- 撰写程序，完成后按[File]菜单→[Save] 命令或按下快捷键“Ctrl+S”存盘，如TEST.ASM 文件名；
- 如需建立多个源程序文件，可重复上面两步操作；
- 打开步骤一中建立的项目源文件，使用Include伪指令将这些文件包含进来；

步骤三：编译调试项目

- [Debug]菜单→[Build]命令或“F7”进行编译；
- 设置编译选项Code Option各项的内容，完成后点击OK确认；
- 系统将会对项目中的所有源程序文件执行编译动作−如果程序中有错误，编译器会将错误提示显示在输出信息栏，用户只要在错误信息行连按两次，系统将会提示错误发生的位置并且打开此错误所在的源程序文件，可直接修改程序及存储文件；
- 如果所有程序文件都没有错误，系统会产生一个执行文件并且载入到ICE中，准备仿真及除错；
- [Debug]菜单→[Go]命令或“F5”运行程序；

重复上述步骤直到没有错误。

步骤四：烧写 OTP 芯片

- [开始]→[程序]→[Sonix]→[M2IDE_V119]→MPIII_WtV106.exe开启烧录界面；
- 选择MCU型号及烧录文件*.SN8，该.SN8档在程序编译后自动生成，且和源程序存放在同一目录下；
- .SN8下载成功后，MPIII_WtV106.exe右面的信息框会给出下载成功的提示；
- 点击自动烧录，烧写OTP 芯片；

# 2.2 菜单--文件/编辑/视图/调试/辅助/窗口/帮助选项

## 2.2.1 启动M2IDE系统

安装完成 M2IDE 软件后，双击桌面上的快捷图标或单击开始/程序/Sonix/M2IDE_V119/M2Asm119.exe，即可进入集成开发环境。如图 2-1 所示：

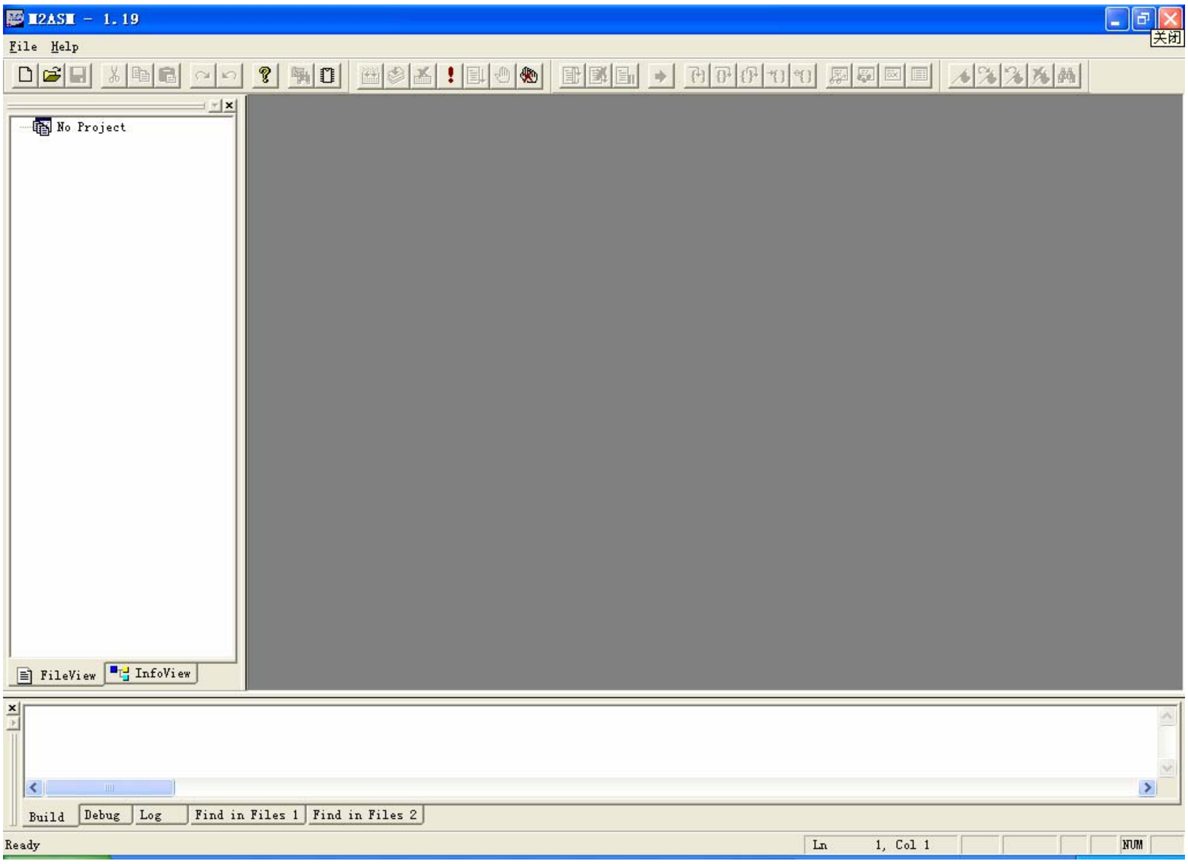



图 2-1 集成开发环境

如果是第一次使用 M2Asm，M2Asm 会自动打开 SN8Readme.txt 并显示欢迎对话框。

M2IDE 编译器每升级一次，会将新版编译器的信息增添在 SN8Readme.txt 里面，在使用编译器前，用户需要仔细阅读这份说明。

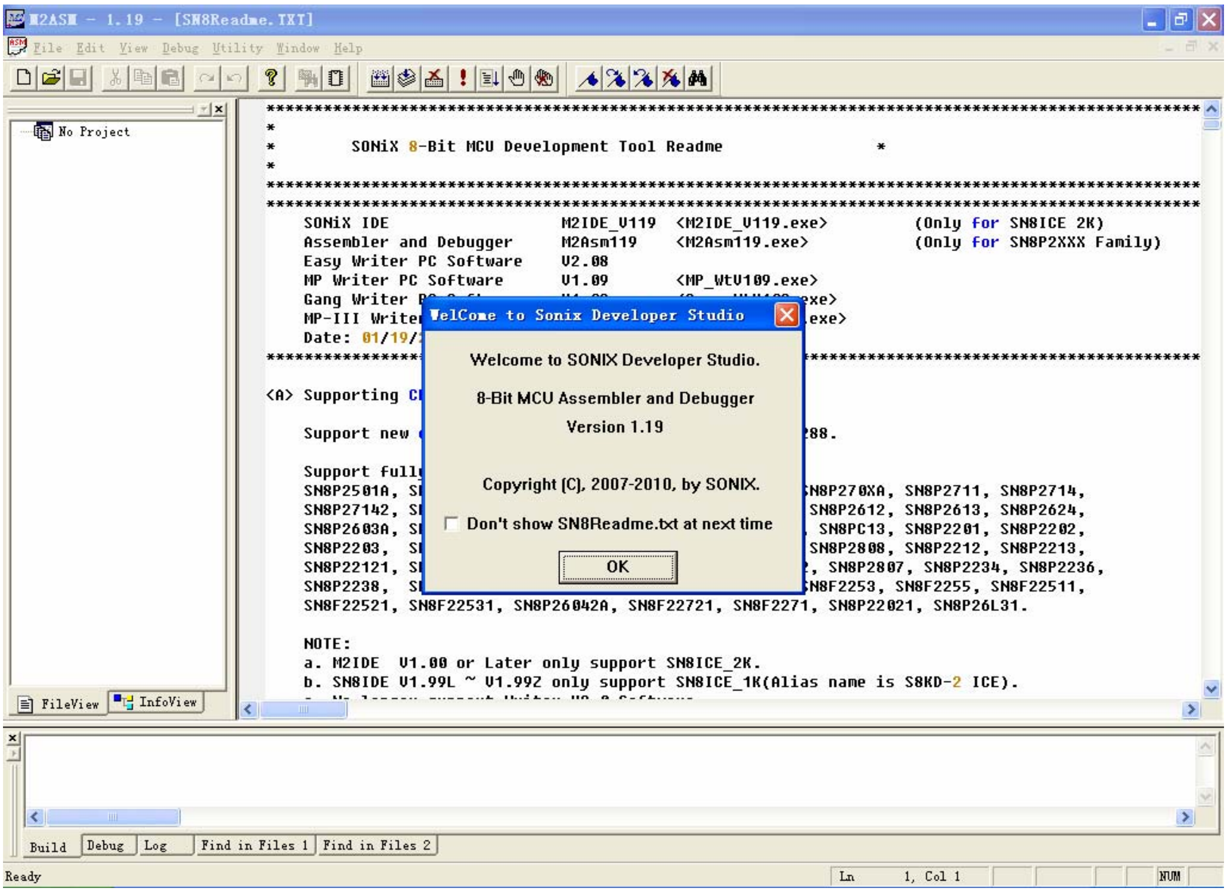



图 2-2 首次打开 M2Asm 界面

若不希望以后每次打开时都显示此画面，可以将 Don‘t show SN8Readme.txt at next time 前面的小方框选中，并点击 OK 即可。

另外用户还可以通过以下途径打开 SN8Readme.txt：
1）执行：开始/程序/Sonix/M2IDE_V119/SN8Readme.txt；
2）编译器安装目录下察看该文件。

## 2.2.2 M2IDE界面

如图所示，图 2-3 为编辑状态，图 2-4 为调试状态。各种调试工具、命令菜单、运行状态等都集中到此开发环境中。

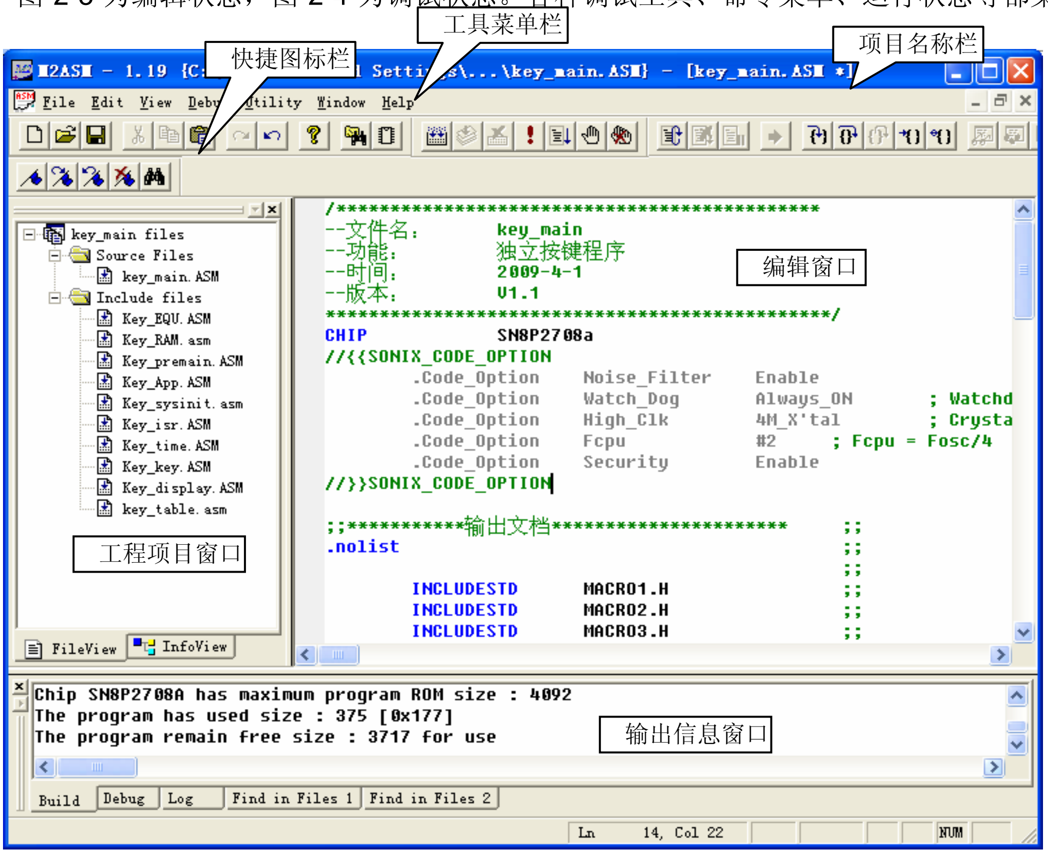



图 2-3 编辑状态操作界面

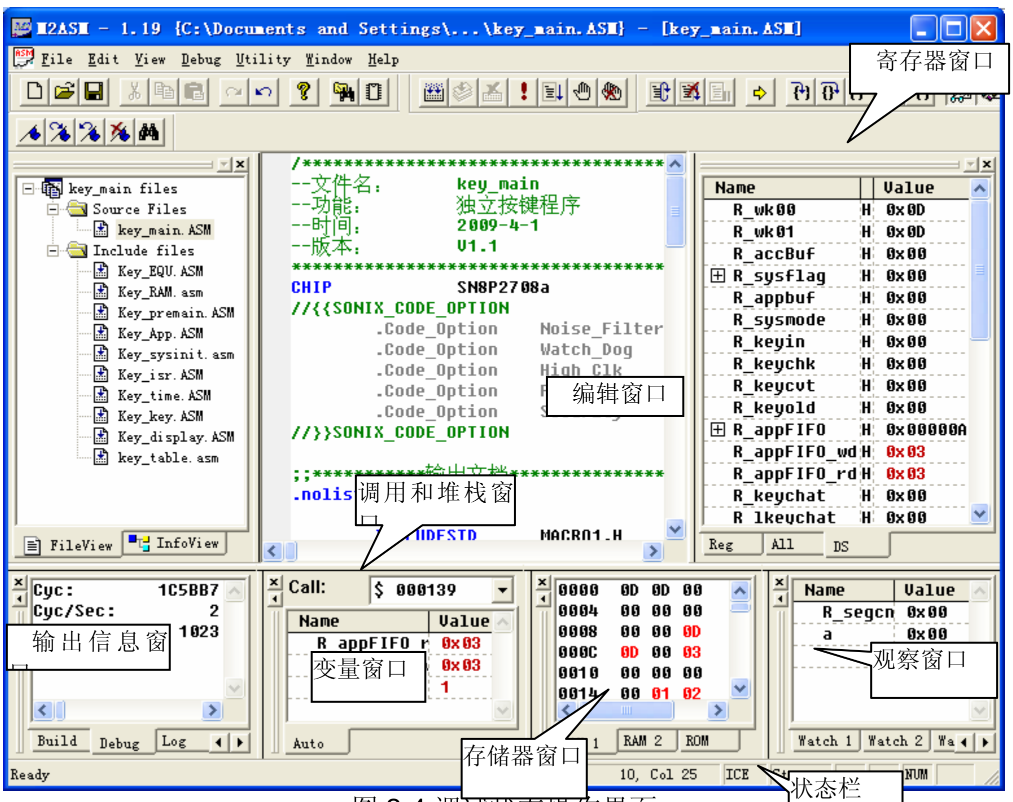



图 2-4 调试状态操作界面

下面简要说明各种窗口的作用和功能。

**菜单栏**

菜单栏提供了各种操作菜单，例如工程建立、编辑操作、程序编译、连接、调试等。

**快捷图标栏**

快捷图标栏为使用较频繁的工具提供了一种快捷操作方式，例如新建文件、编译、运行等。

**项目名称栏**

项目名称栏显示当前项目的名称及存储路径。

**编辑窗**

编辑窗口用于进行程序的输入和修改。

**状态栏**

显示当前调试状态，包括运行、停止状态，程序当前所在的行等信息。

状态栏所显示的信息可能在程序编译及调试阶段有所帮助。例如在程序编辑时，指示行与列的值，Ln后面的数据指示当前光标所处位置的行，Col指示当前光标所处位置的列 (状态栏最右边的两个字段)会显示光标当前的位置。

编译时指示Simu…，运行时指示ICE Run…，运行停止时指示ICE Stop。

用户可以通过在工具栏或状态栏空白处单击右键，选择Staus Bar来显示和关闭状态栏。

**工程项目窗**

工程项目窗分为两页，Fileview 页显示当前项目组的信息；Infoview 页显示当前工程中的全部标号，并可定标所在的程序行，方便用户调试程序。

**输出信息窗**

输出信息窗分为 Build、Debug、Log、Find in Files1、Find in Files2 等五页。

Build 页显示工程建立过程中的信息。包括编译、连接、校验、代码大小、警告、错误等信息。

> 注意：
> Checksum：校验和，用户可以通过烧录器进行 checksum 校验，用于判断烧录进 IC 的资料是否正确。

使用 SONiX 的 IDE 进行编译，编译成功后，在 IDE 的输出信息栏将给出程序代码的一些信息。

如果用户启动加密选项“Enable Security”，编译后生成两个 Checksum 值 --- EPROM Checksum 和 Security Checksum；否则只生成一个 Checksum --- EPROM Checksum；

EPROM Checksum 是指用户源代码的 Checksum 值；

Security Checksum 是指在 SONiX 的加密机制下用户代码的 Checksum 值；

Debug 页显示程序执行指令数，执行时间等信息。Find in Files 页用于显示查找的指定文件中某字符串的全部列表。Cyc：表示所执行指令的指令周期；

Cyc/Sec：程序跑起来后再停下来，根据该数据，可以大概估算出程序的执行速率；

Trace：保存已经执行的最近 1024 个指令的内容，按 Ctrl+Shift+F11,可以后退程序，并观察。

**变量窗**

变量窗是以 AUTO 的状态来显示运行过程中被改变的当前变量。它和观察窗一样都显示变量的当前值，其显示格式也完全相同，但是变量窗里面无法自己设定。在调试状态该窗口才会出现，用户可以通过单击工具栏中的按钮，或者通过 View 菜单，选择 Debug Windows 再弹出下拉菜单后，选择 Variables 项来打开或者关闭该窗口（若原来没有打开该窗口，执行该命令可以打开该窗口；若原来已经打开该窗口，执行该命令就可以关闭该窗口）。一般地，该窗口中自动显示当前指令和接下来两条指令影响的变量的值，方便用户实时跟踪当前程序中的变量。

**调用和堆栈窗**

堆栈窗口显示堆栈的使用情况和入栈函数，在此可以判断程序调用是否正确。

该窗口所显示的地址为当前程序运行位置所被调用的入口地址，通过选择该地址可以查看被那个 Call 调用。

堆栈是从 0 开始往上增加层次，每次入栈操作会使堆栈层数加 1（例如 CALL 指令或中断相应，而每次出栈操作则使堆栈层数减 1（RET 或 RETI 指令）。堆栈窗中显示当前程序堆栈的入口地址，最外层的堆栈显示在最下层，没有堆栈时显示$00000。

调用和堆栈窗显示当前运行状态下，堆栈的使用情况和入栈的子程序或中断子程序的地址。用户可以根据显示来判断程序的调用状况，进而判断当前的状态是否正确，有没有调用出错。

**存储器窗口**

存储器窗口分别显示片内数据存储器和程序存储器当前的数据和代码。大部分MCU的存储器窗口都有3页，RAM1，RAM2和ROM页，也有一些例外，如USB系列MCU具有USB FIFO 0，USB FIFO 1，SN8P2308具有LCD页等。

在调试状态该窗口才会出现，用户可以通过单击工具栏中的 Memory 按钮，或者通过 View 菜单，选择 Debug Windows 再弹出下拉菜单后，选择 Memory 的来打开或者关闭该窗口。

**观察窗**

观察窗用于监视用户定义的变量。为方便观察，可以将变量分类放入 4 个不同的观察页 Watch1、Watch2、Watch3、Watch4。

观察窗用于监视用户定义的变量，即显示数据存储器 RAM 中寄存器的内容。为方便观察，可以将变量分类放入 4 个不同的观察页 Watch1、Watch2、Watch3、Watch4。

使用以下两种方法添加要监视的变量：

第一种方法是用鼠标单击观察框的 watch 页 Name 下的空白处，待出现光标进入编辑状态后即可以添加变量。

第二种方法是在编辑窗口中双击要监视的变量，然后拷贝到观察窗口当中，就可以看到变量的当前值。

第三种方法是选中变量，单击鼠标右键，再弹出的菜单中选择 Add to Watch of xx，即可将变量添加到 watch 中。

**寄存器窗口**

寄存器窗口显示寄存器及 DS 定义的变量当前的值。用户可以通过单击工具栏中的 Register 按钮，或者通过 View 菜单，选择 Debug Windows 再弹出下拉菜单后，选择 Regirster 的来打开或者关闭该窗口。此窗口分 3 页显示，分别是 Reg、All 和 DS，Reg 显示的为系统最常用的寄存器；All 中显示所有的系统寄存器；DS 中显示用户定义的寄存器变量。如图 2-5 所示。

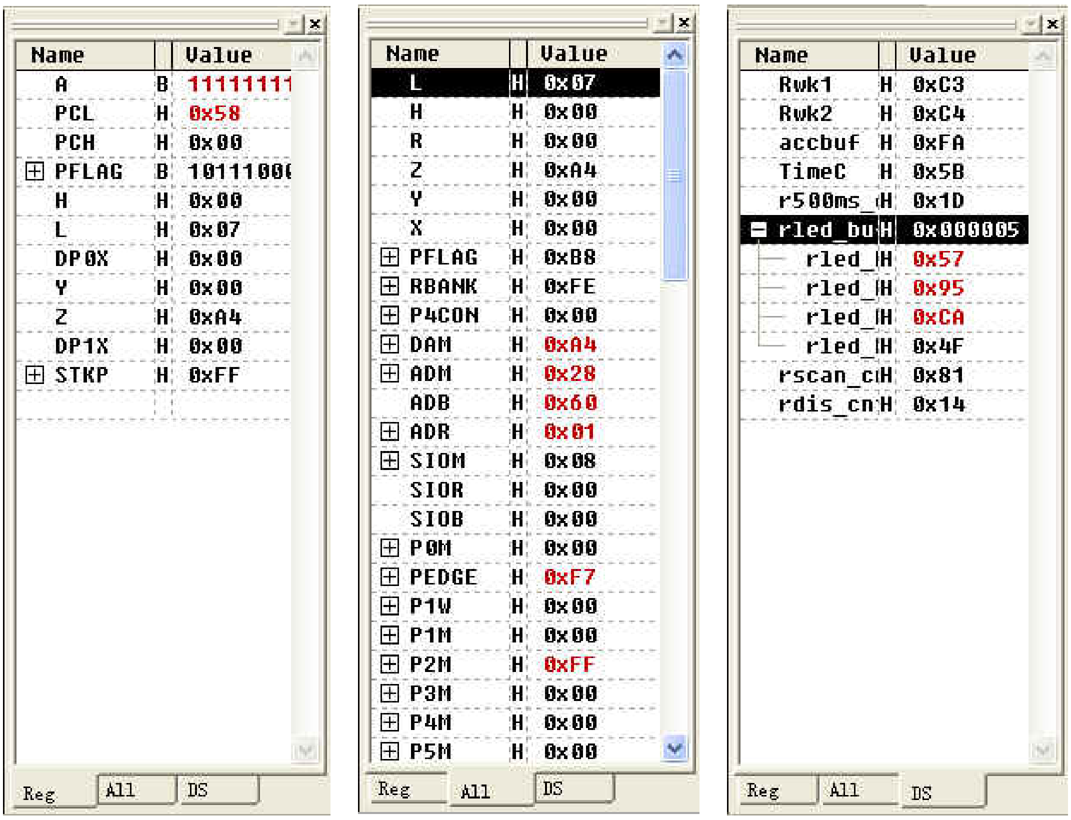



图 2-5 寄存器窗口

下面详细介绍各菜单的作用。

### 2.2.3 文件菜单（File）

File 菜单主要用于新建一个文件/工程，或打开一个文件或工程。如图 2-6 所示。

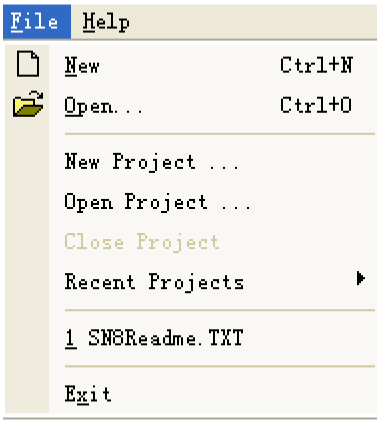



图 2-6 文件菜单

- New：建立新文件；
- Open： 打开已有文件；
- New Project：新建一个工程文件；
- Open Project：打开一个已有工程；
- Close Project：关闭一个工程；
- Recent Project：打开最近打开过的工程文件；
- 1~9：打开最近使用的源文件或文本文件；
- Exit：关闭/退出并提示保存文件。

当已打开一个源文件时，还会有Close，Save，Save as…等选项。

- Save： 保存；

| 注意：<br>1 菜单栏和各菜单中的下划线表示该菜单的快捷键字母，用户按 Alt+相应的字母即可打开该菜单；<br>2 该菜单打开后，直接按相应的字母即可执行相应的功能。 |
|---|

举例如下：

如要从编译界面打开一个.ASM文件，步骤如下：

步骤1：打开M2IDE（如图2-7）；

步骤2：在编译器界面下按“Alt+F” (此时Caps Lock打开与否不影响)（执行后如图2-8）。

步骤3：若该文件新近打开过，在1~9中有显示，则直接按相应的字母；若没有，则按键盘上的“O”键（执行后如图2-9）；

步骤4：在电脑中选择需要打开的文件即可。

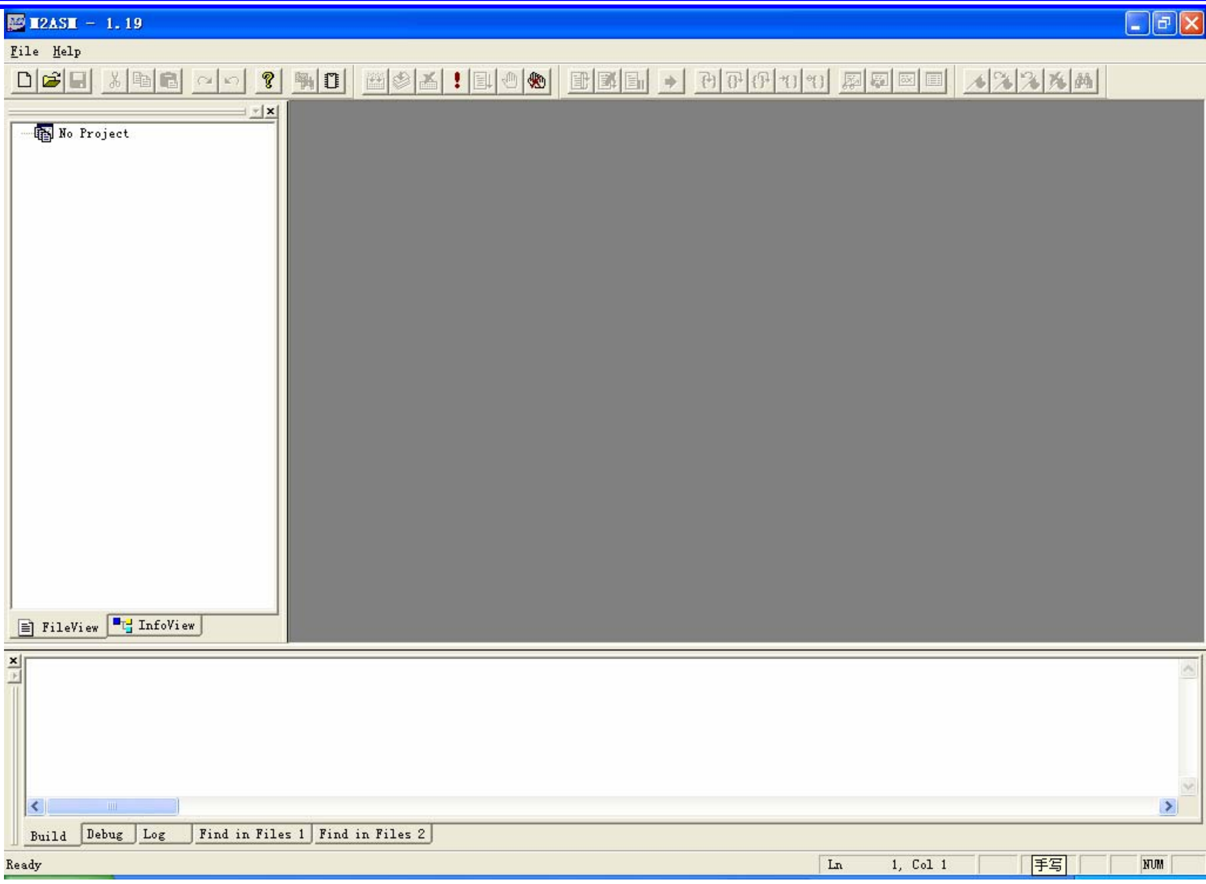



图 2-7 M2IDE 界面

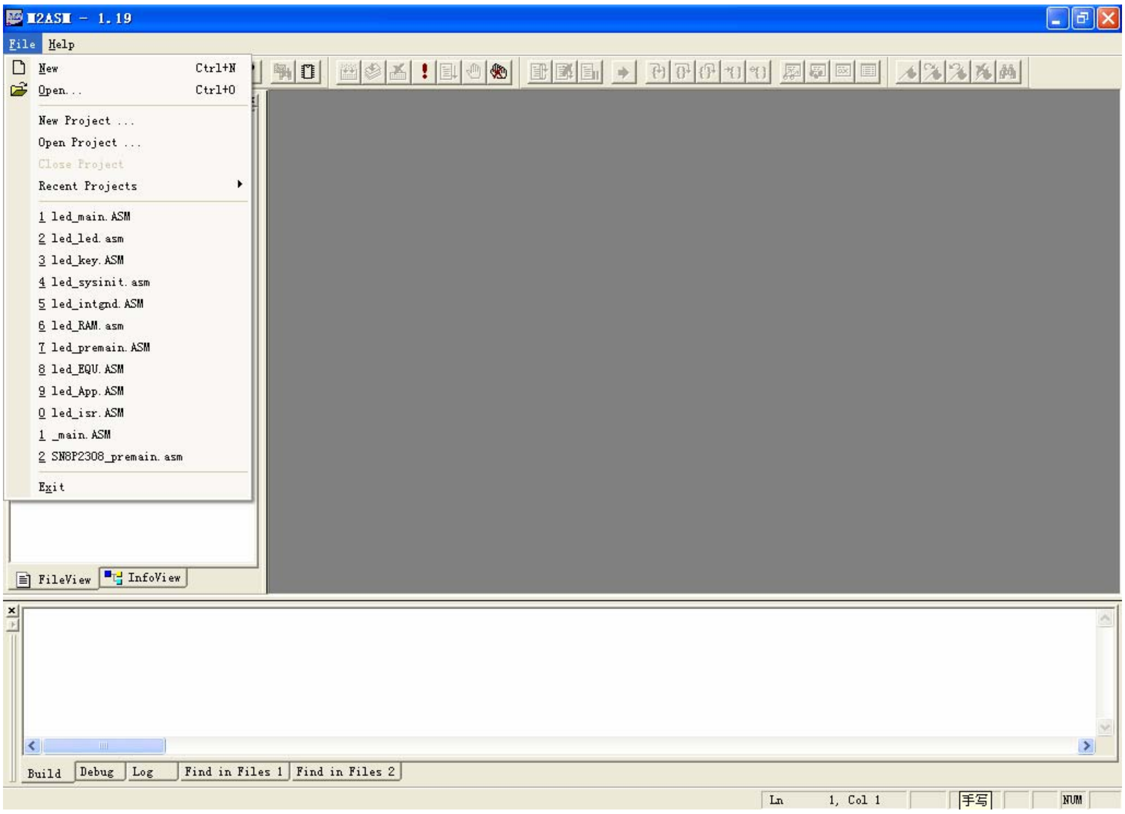



图 2-8 打开 file 菜单

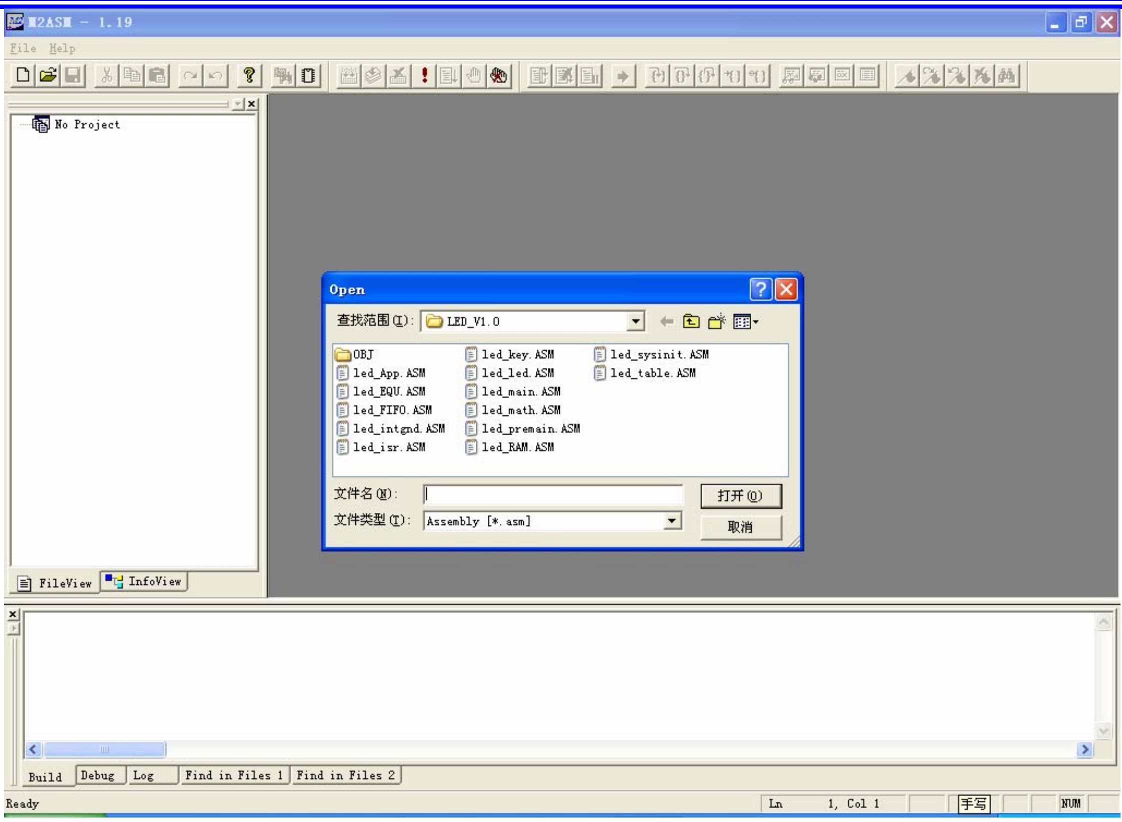



图 2-9 选择文件名称

这样即可实现迅速快捷得打开一个文件了。

此外，用户在打开 File 菜单后，也可以使用键盘上的上下键来选择，启动所需要执行的功能，用户在用上下键选择时，被选中的项会被高亮表示。如图 2-10：

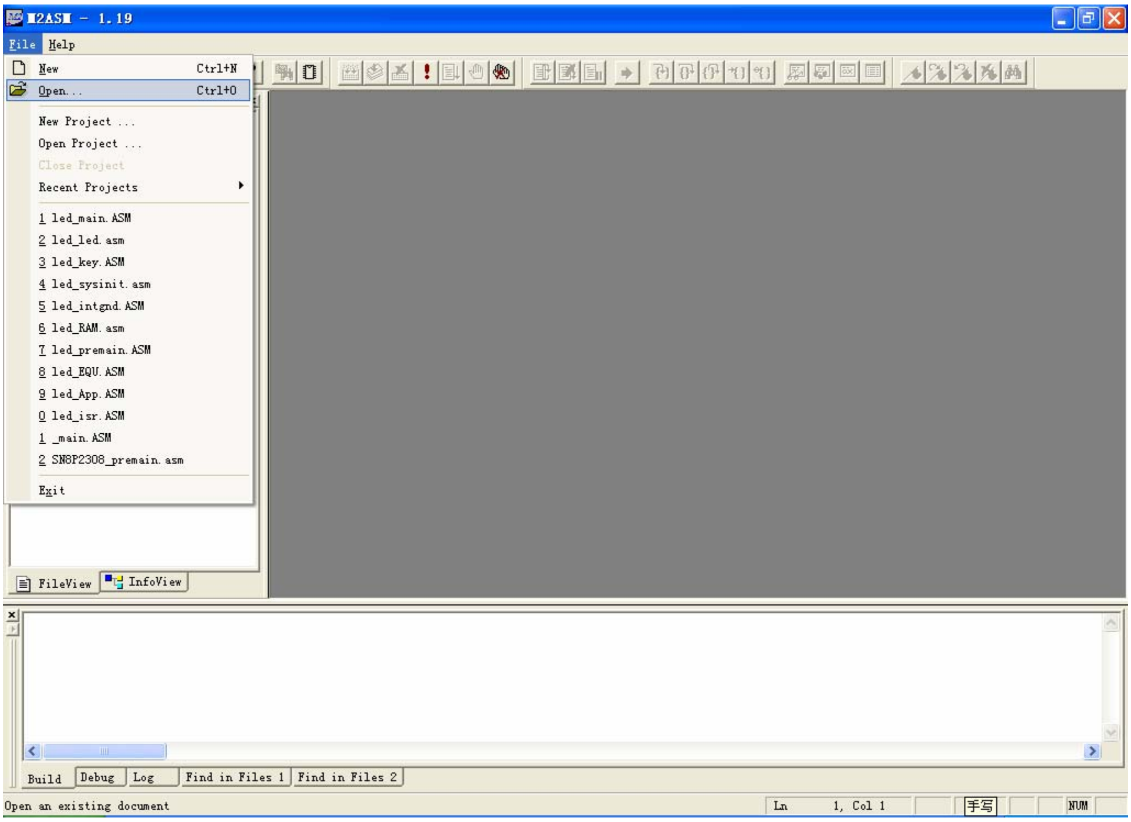



图 2-10 使用上下键选择

相比于使用鼠标，这种使用快捷键操作的方法更为方便和快捷。

## 2.2.4 编辑菜单（Edit）

Edit 菜单主要完成对程序代码的取消，重做，剪切，复制，粘贴，全选，查找及对标签操作等。

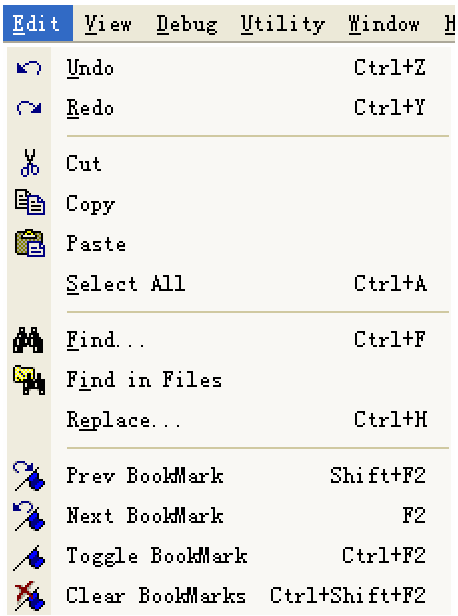



图 2-11 编辑菜单

- Undo ：撤销上一步操作；
- Redo ：恢复上一步操作；
- Cut：剪切；
- Copy：复制；
- Paste：粘贴；
- Select All：全选；
- Find：在当前文件中查找；
- Find in Files：在所有文件中查找；
- Replace：替换；
- Prev BookMark：前一个标签；
- Next BookMark：下一个标签；
- Toggle BookMark：创建一个标签，再次按下撤销当前光标所在行标签；
- Clear BookMarks：清除所有标签。

下面举例说明如何使用该菜单下的查找命令。

**当前文件查找：**

如在当前文件（如图2-12）中查找一个用户命名的寄存器R_keyin。步骤如下：

步骤1：执行Edit---〉Find命令，或按快捷图标进行查找，弹出如图2-13所示对话框；

步骤2：此时用户可以根据需要一个一个查找（Find Next），或者直接将所有具有R_keyin的行标识起来（MaskAll），再配合Prev BookMark，Next BookMark，Toggle BookMark，Clear BookMarks等操作查找。

执行MaskAll后如图2-14所示。

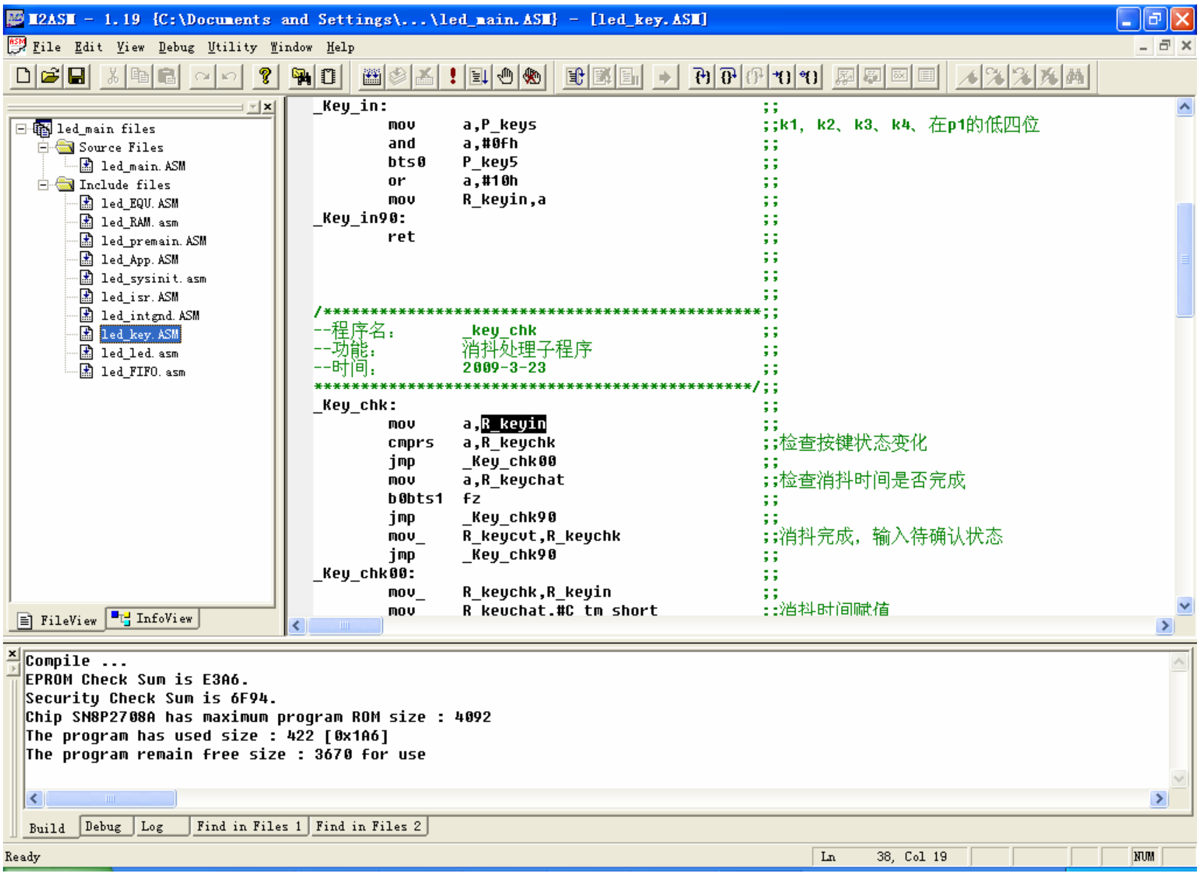



图2-12 当前文档查找寄存器

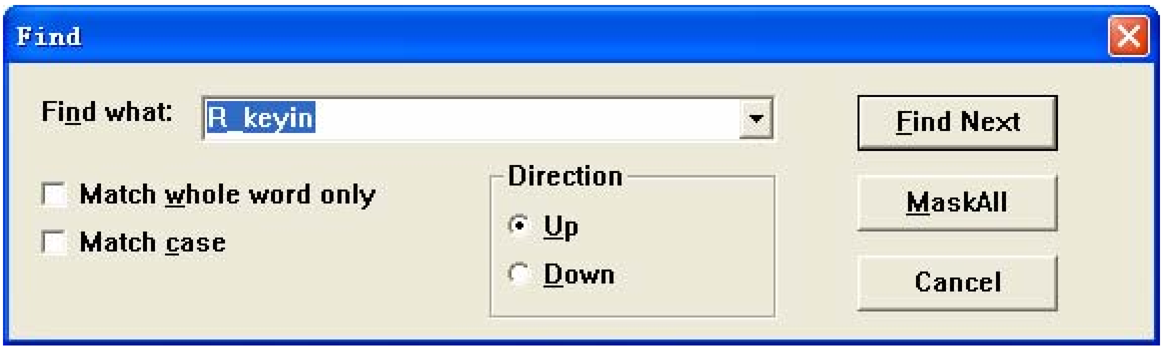



图 2-13 当前文件寄存器查找信息框

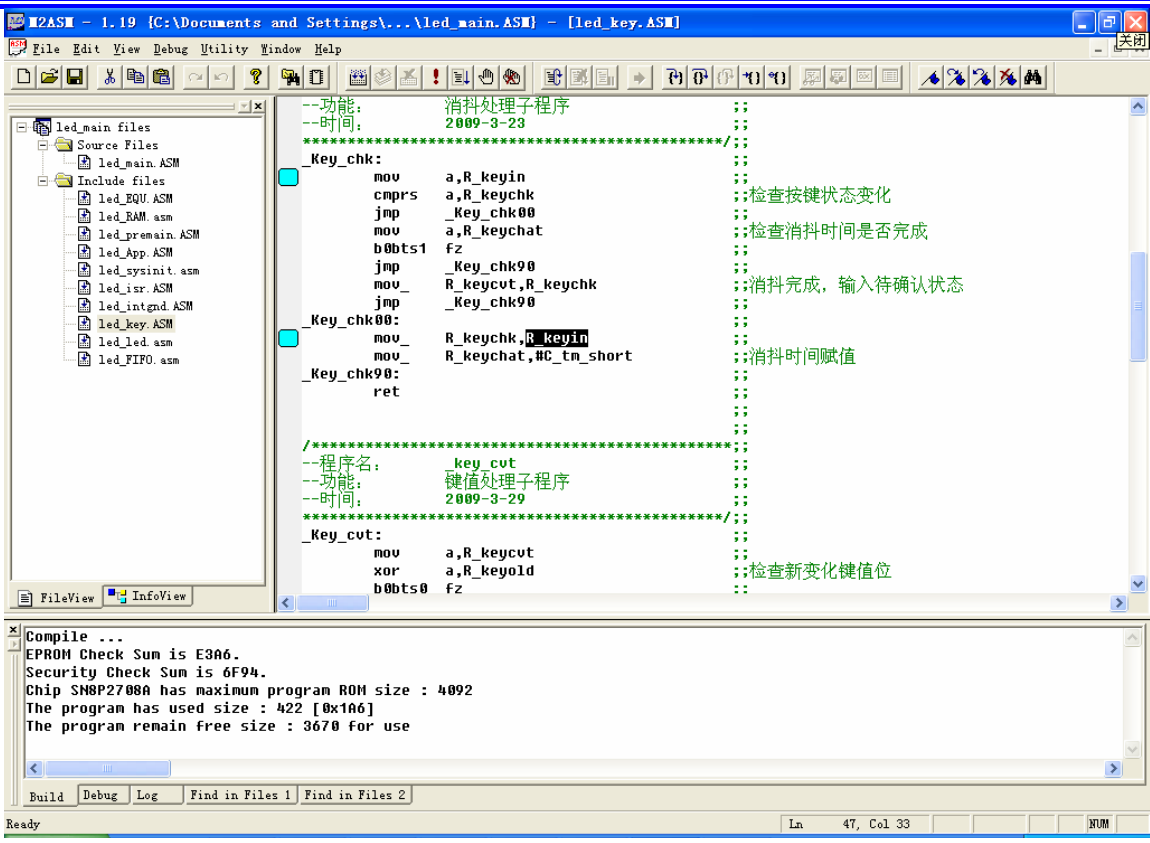



图 2-14 执行 MaskAll 后编译器界面

整个文件查找：

Find in Files：

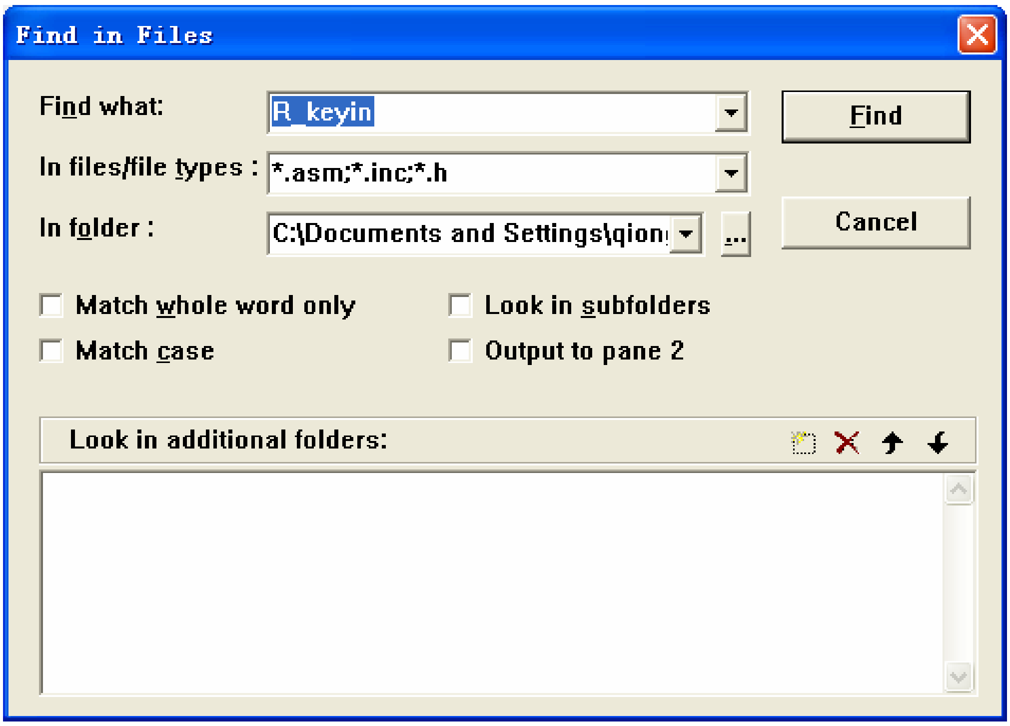



图 2-15 整个文件查找信息框

Replace：

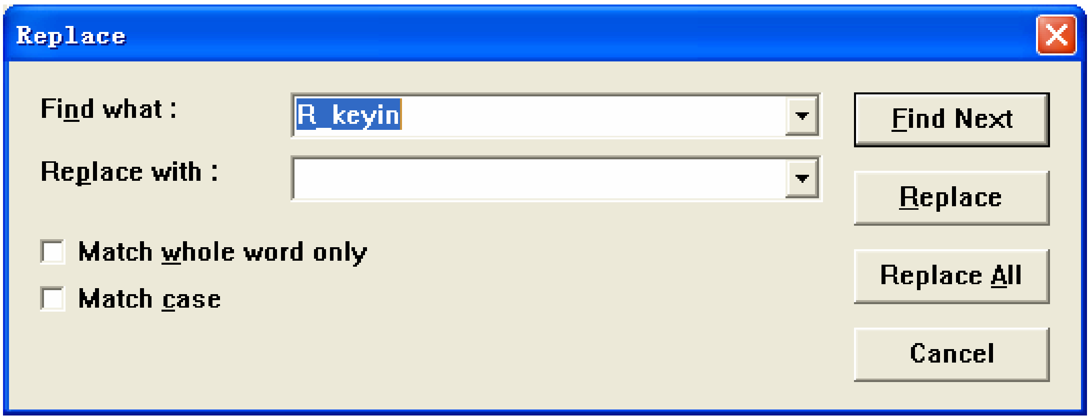



图 2-16 替换信息框

注意：
在 Find，Find in files，Replace 弹出的对话框中均有两个选项：
Match whole word only 和 Match case
其中
Match whole word only 表示：与所查找字母组合完全匹配且独立，否则不显示结果；
Match case 表示：与所查找字母组合相同。

## 2.2.5 视图菜单（View）

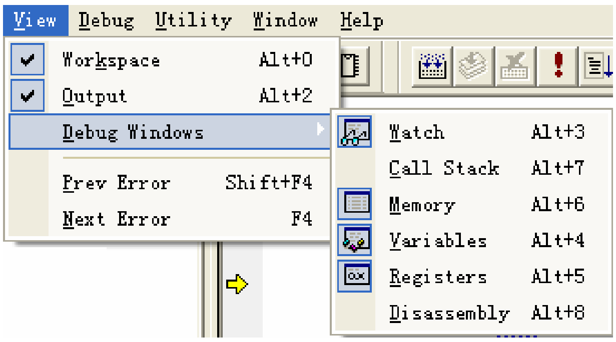



图 2-17 视图菜单

- Workspace：选中则显示工程项目窗口，默认选中状态；
- Output： 选中则显示输出信息窗口，默认选中状态；
- Debug Windows：调试窗口（其子菜单在程序编译并运行一次后才可以正常显示，其它情况下为灰色）；
- Watch： 观察窗口，运行程序后默认打开；
- Call Stack：堆栈调用级数窗口，保留；
- Memory：存储器窗口，运行程序后默认打开；
- Variables：变量窗口，运行程序后默认打开；
- Registers：寄存器窗口，运行程序后默认打开；
- Disassembly：汇编窗口，保留；
- Prev Error：编译不通过时编译器会有报错提示，执行此命令，察看前一个错误；
- Next Error：编译不通过时编译器会有报错提示，执行此命令，察看下一个错误。

## 2.2.6 调试菜单（Debug）




图 2-18 调试菜单

- Build：编译程序；
- Rebuild All：重新编译所有（该选项在建立工程时有效）；

停止编译（通常状态下为灰色）；

- 停止编译，该图标在通常情况下为灰色（常用图标栏中有）；
- Download…：执行该命令，程序将自动下载到仿真器的FPGA中；
- Reset：复位程序 ；
- Go：运行程序；
- Break：暂停当前运行的程序；
- Stop Debugging：停止调试，返回编译后状态；
- Cursor：光标，显示下一状态，指向当前指令（常用图标栏中有）；
- Single：快捷图标栏也叫Step Into命令，单步运行。一次只执行一条指令就停止，但是如果所执行的指令是CALL程序指令时，则会进入此子程序并且停在子程序中的第一条指令处；
- Step Over：跳过函数运行。一次只执行一条指令就停止，但是如果所执行的指令是call 程序指令时，则不会进入子程序，而是停留在call 指令后的下一条指令处，此子程序中所有的指令会被执行，而且寄存器与状态寄存器的内容也根据执行的结果做改变。
- Step Out：运行结束。只能在仿真停在子程序之内时被使用，它会执行当前停在的程序行和RET 指令（包括RET 指令）之间所有的指令，然后停留在call指令后的下一条指令处。
- Run to Cursor：运行至光标所在处（有记忆，用户按Ctrl+Shift+F11可以返回上一条指令并察看）。执行此命令可使程序执行到代码窗口中光标所在的位置。这相当于把光标所在的位置作为一个临时的断点。
- PC to Cursor：指向光标所在处（没有记忆）。执行此命令可以使程序计数器指向当前光标所在的行，注意这里只修改程序计数器PC，并不执行任何的指令。
- Insert/Remove Breakpoint：插入一个断点/取消一个断点；
- Breakpoints….：断点设置；
- Remove All Breakpoints：取消所有断点；

●Fill RAM：给寄存器里面写入数据；
● Animate Single：单步自动运行。如果遇到call指令，会进入子程序，并开始执行call里面程序的第一条指令；
● Animate StepOver：跳过函数自动运行。如果遇到call指令，不会进入子程序，而是接着执行call 指令后的下一条指令处，此子程序中所有的指令会被执行，而且寄存器与状态寄存器的内容也根据执行的结果做改变；
● Prev Single Trace：跟踪前一条语句；
● Pre Trace：跟踪至前64条指令；
● NextTrace：跟踪到下一句；

注意：
Step out 命令只能被使用在当前的停止点在子程序内的情况下，否则会发生无法预期的结果。

Breakpoints….：
使用此命令，必须先设置一个断点，没有设置断点时执行该命令时对话框如下：

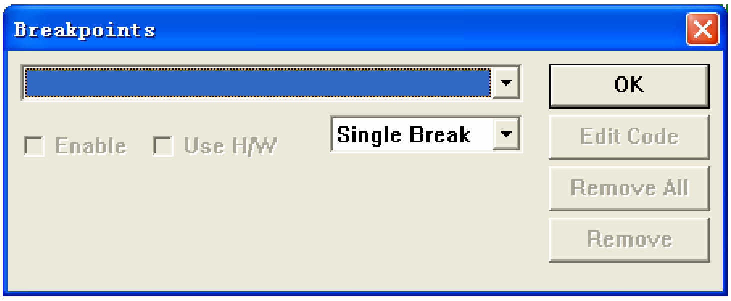



图 2-19 没有设置断点时执行 Breakpoints 对话框

设置断点后，执行该命令后对话框如下：

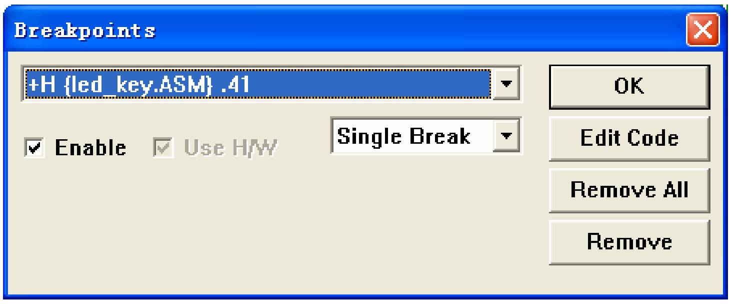



图 2-20 设置断点后执行 Breakpoints 对话框

文件中若设置多个断点，可以选择各断点（如图2-21），并可以选择触发的方式（如图2-22）。

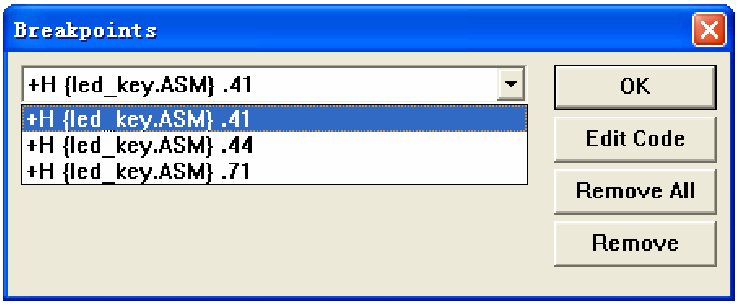



图 2-21 设置多个断点时对话框

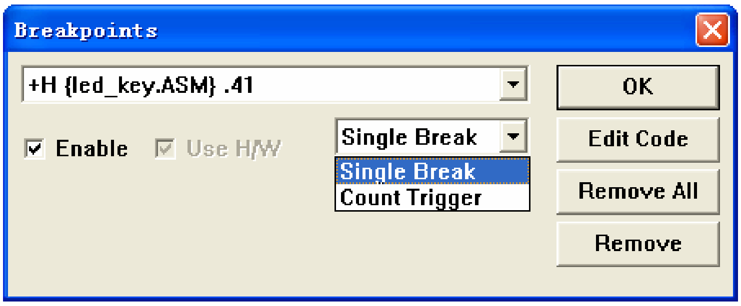



图 2-22 选择触发方式对话框

设置触发次数后显示如图所示，以设置 40 次为例：

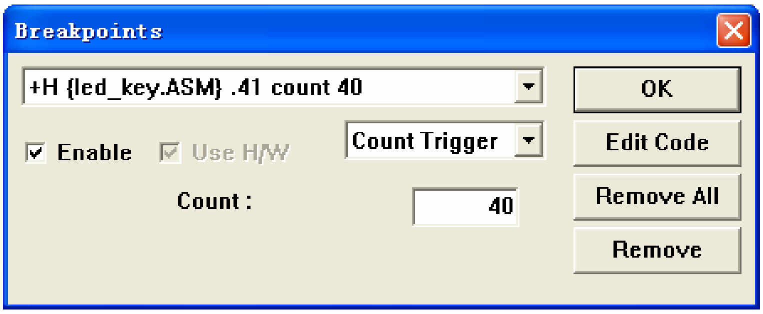



图 2-23 设置触发次数后对话框

Download……：

下载后程序可以脱离计算机环境自由运行。使用该命令后将弹出如图所示的文件选择对话框，选择要下载的.SN8 文件，单击打开，弹出图所示的下载成功提示框，提示用户下载文件成功，MCU 正在自由运行。

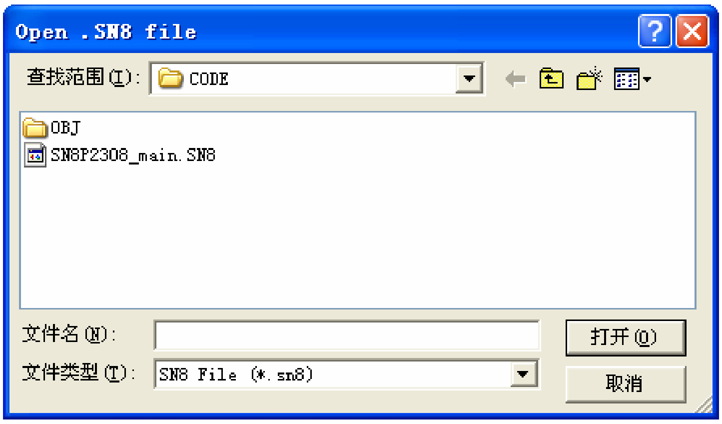



图 2-24 下载程序到 ICE 时对话框

选择需要下载的.SN8 档

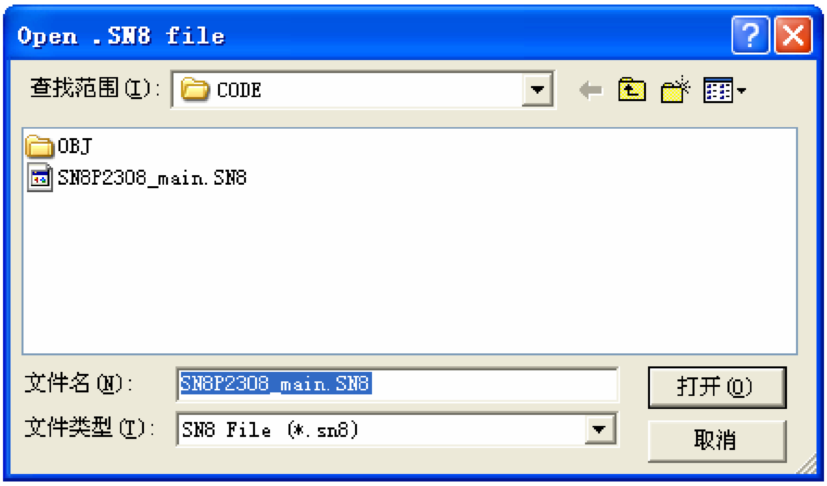



图 2-25 选择需要下载的程序

下载成功后，编译器会给出下载成功提示，画面如下：




图 2-26 下载成功提示框

## 2.2.7 应用菜单（Utility）

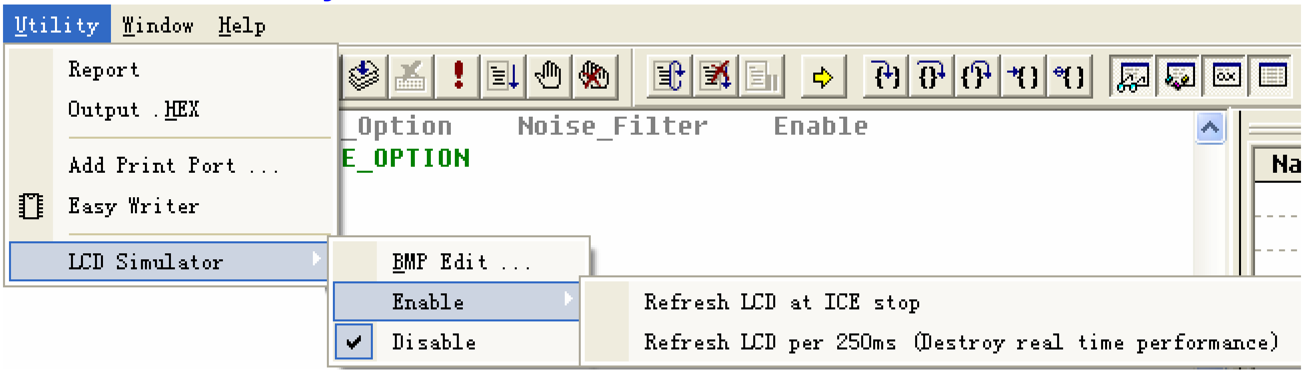



图 2-27 应用菜单

- Report：将.SN8/.BIN文件翻译为.RPT文件；
- Output HEX：将.SN8/.BIN文件翻译为.HEX文件；
- Add Print Port：给仿真器应用添加打印端口；
- Easy Writer：配合Easy Writer 烧录芯片；
- LCD Simulator：仿真LCD功能。

执行相应命令后弹出对话框如下所示：

Report：

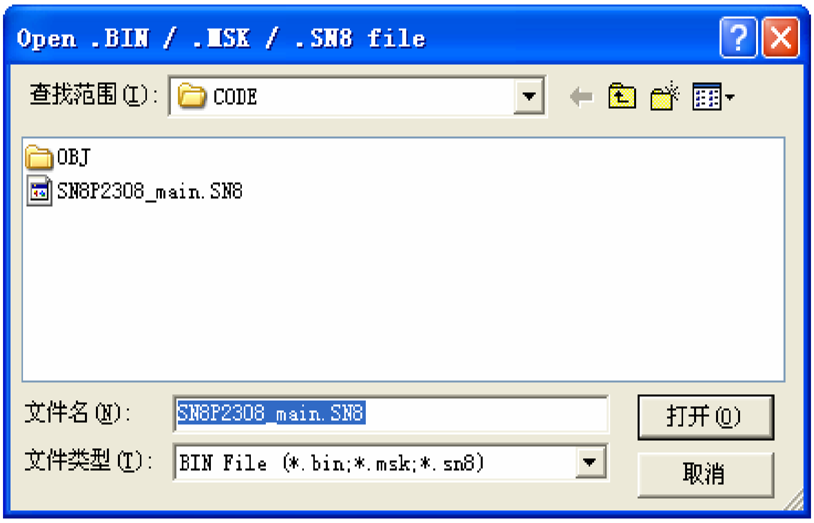



图 2-28 执行 Report 命令后对话框

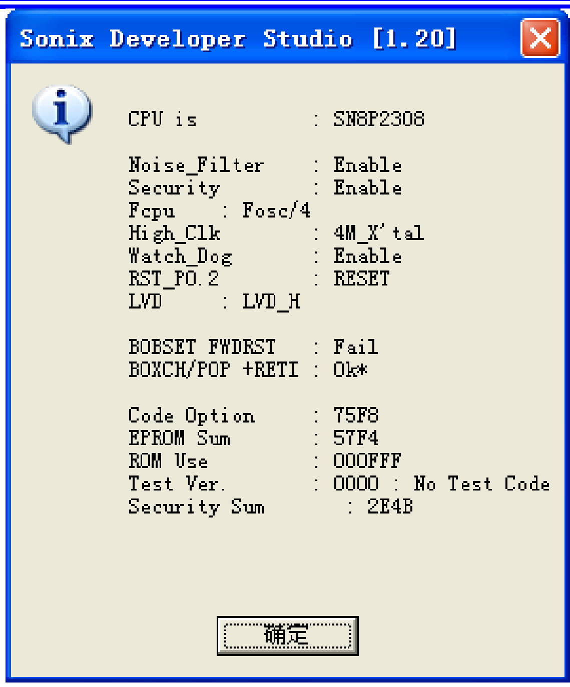



图 2-29 执行 Report 命令后信息框

Output.Hex：
执行该命令后，在源文件目录下，将自动生成一个 SN8P2308_main.HEX 文档。

Easy Writer：
执行该命令后将会弹出如图 2-30 所示信息框，点击确定后可进入 Easy Writer 烧录界面。




图 2-30 执行 Easy Writer 命令后提示框

LCD Simulator：
可参考 **2.4 如何仿真LCD**。

## 2.2.8 窗口菜单（Window）

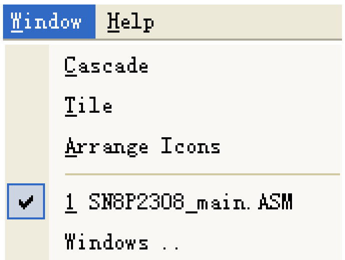



图 2-31 窗口菜单

- Cascade：设置窗口层叠；
- Tile：设置窗口非层叠；
- Arrange Icons：在窗口底部排列图标。

执行各命令后画面如下：
Cascade：设置窗口层叠；

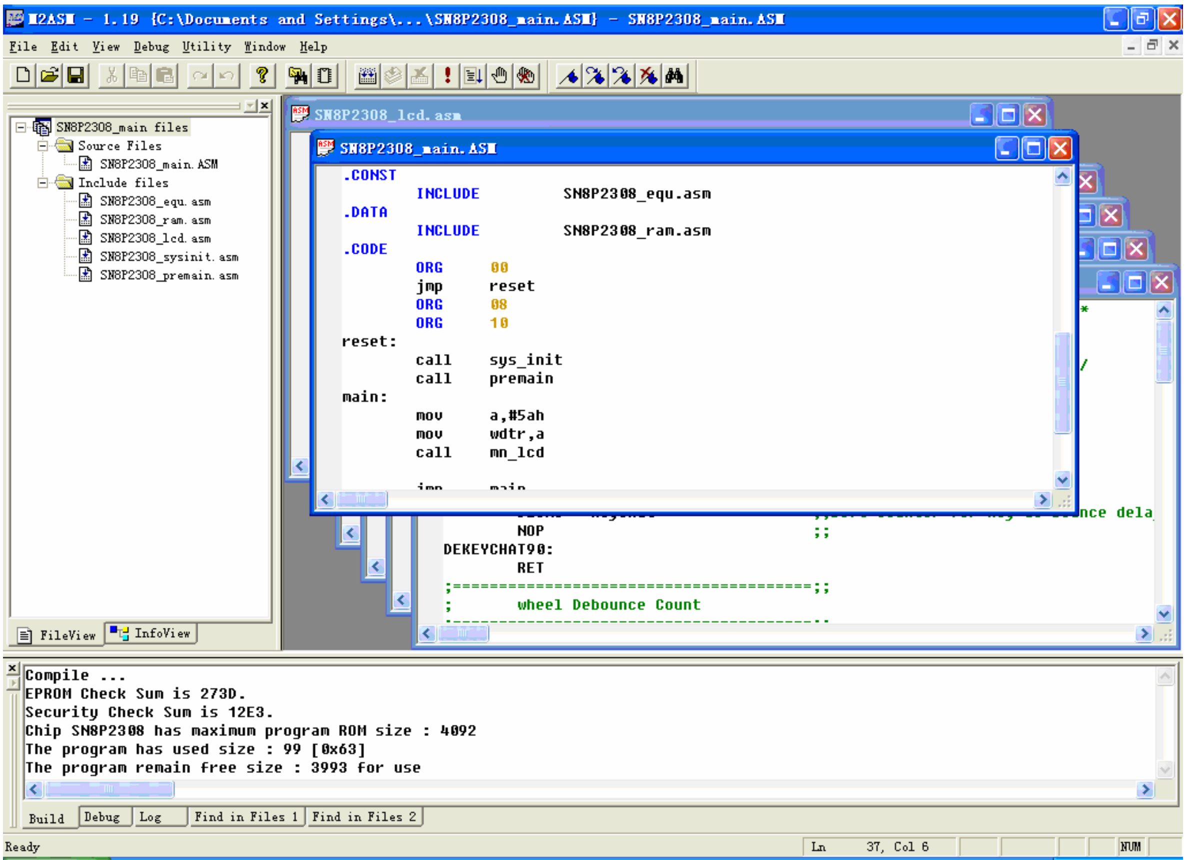



图 2-32 窗口层叠

Tile：设置窗口非层叠；

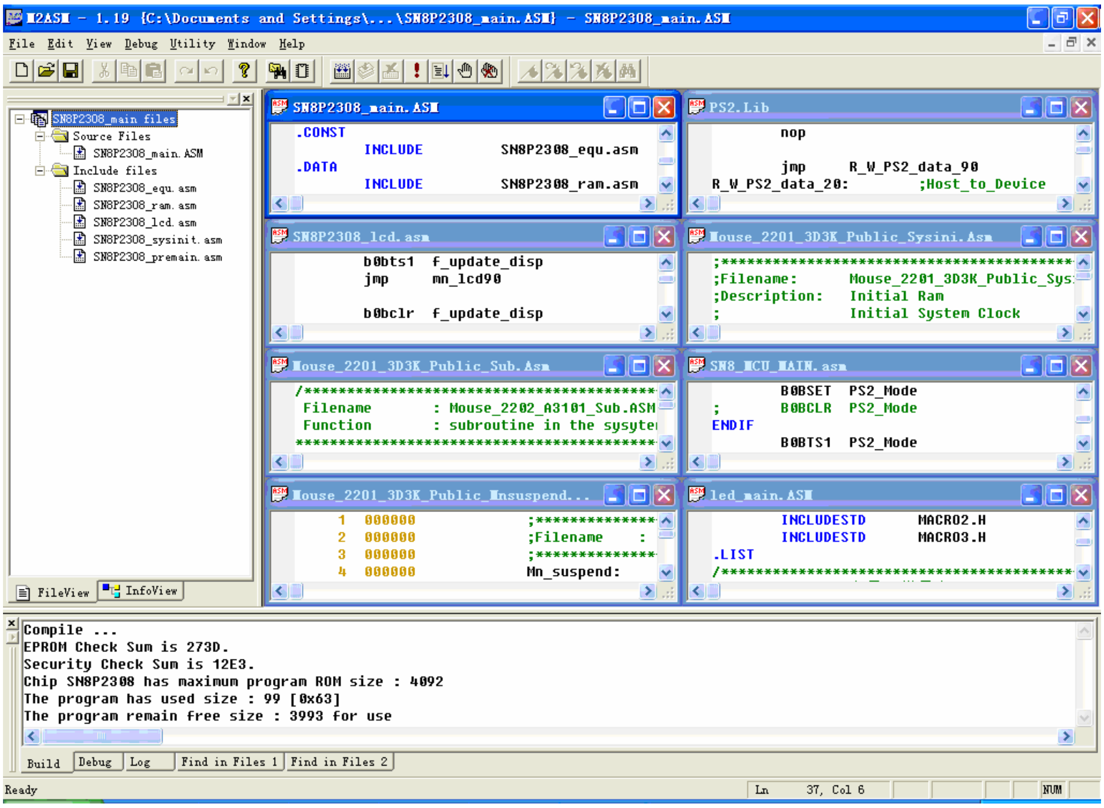



图 2-33 窗口非层叠

Windows：

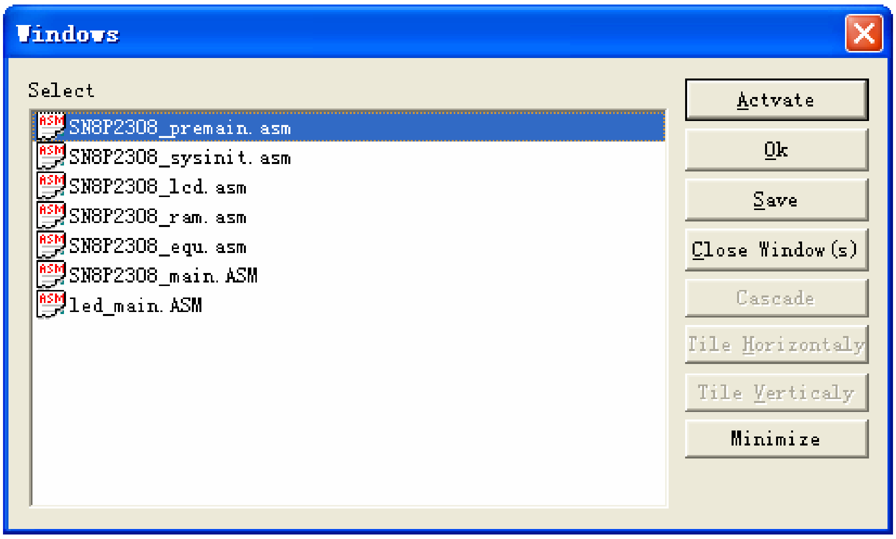



图 2-34 窗口

## 2.2.9 帮助菜单（Help）




图 2-35 帮助菜单

-  About Assembly：帮助。

关于菜单命令、工具及快捷方式，请查看附件 II。

## 2.2.10 窗口管理

| |
|---|
| 注意：<br>1 在 M2IDE 界面下，所有窗口都是可以随处停留的，用户可以在主窗口内把它到处移动。<br>2 所有窗口都是可以通过双击窗口的边缘处将其单独弹出以方便操作（如关闭）和观察，再次双击后，其将重新嵌入编译器界面。<br>3 在工具栏或状态栏空白处单击右键，在弹出的菜单中可以选择是否显示相应的窗口。 |

下面举例分别说明。

1. 在编译器界面下，所有窗口都具有白色的两条竖线或横线标志，这里叫它活动条，如下图：

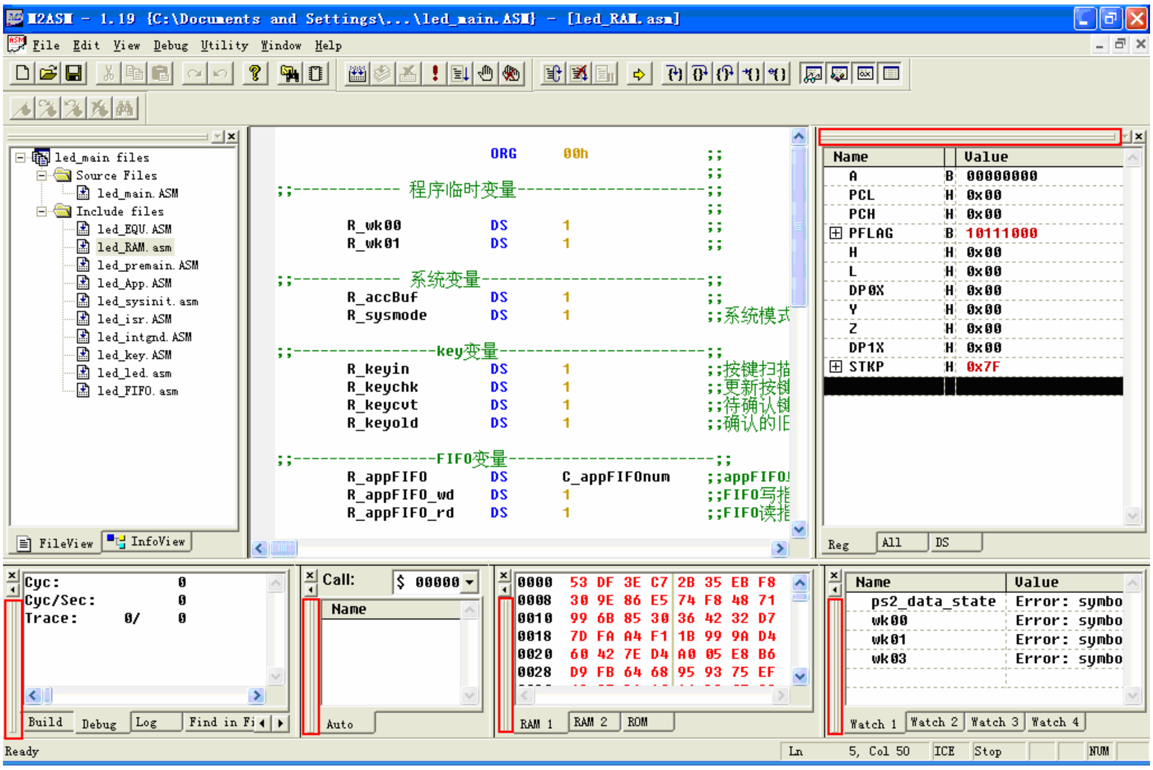



图 2-36 各窗口的活动条显示

单击这些活动条不放并拖动，当该窗口变为一个具有黑色边线的矩形框的时候，即可将其移动到界面的其它地方（如图 2-37 所示）。

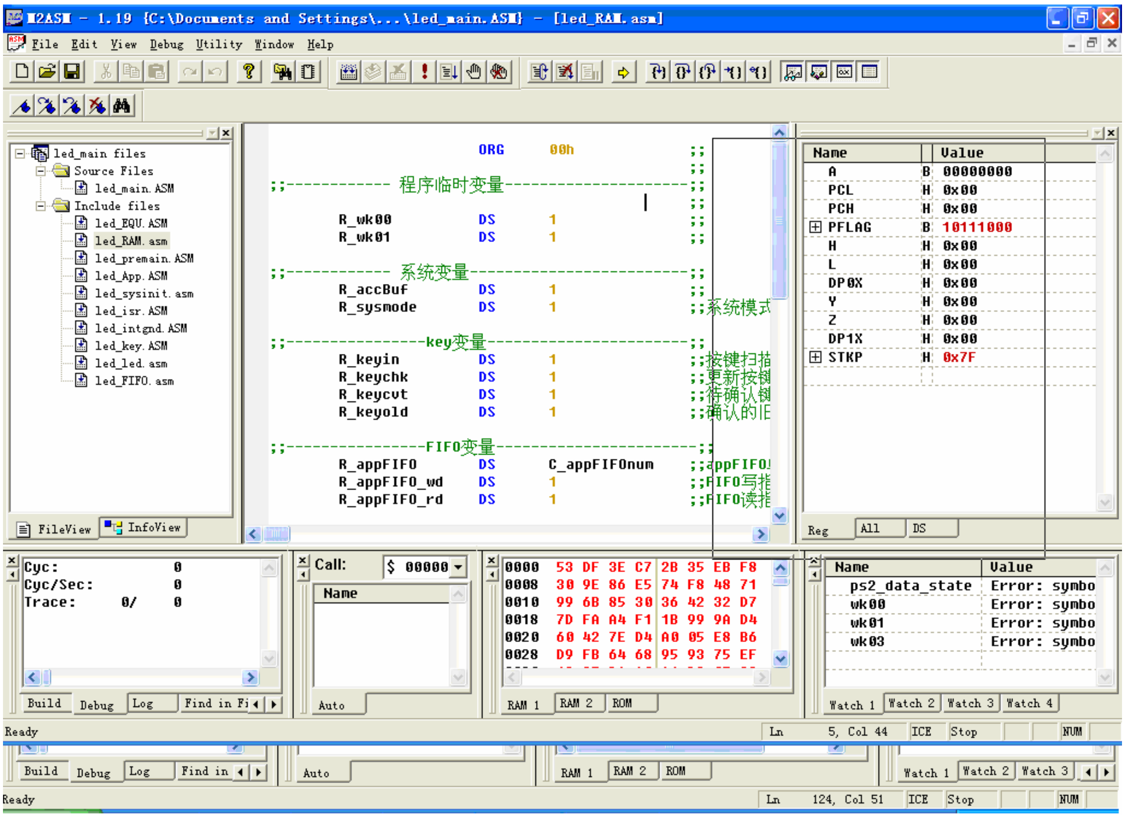



图 2-37 正在移动中的窗口

2. 同样，双击活动条，该窗口即可弹出，如图所示：

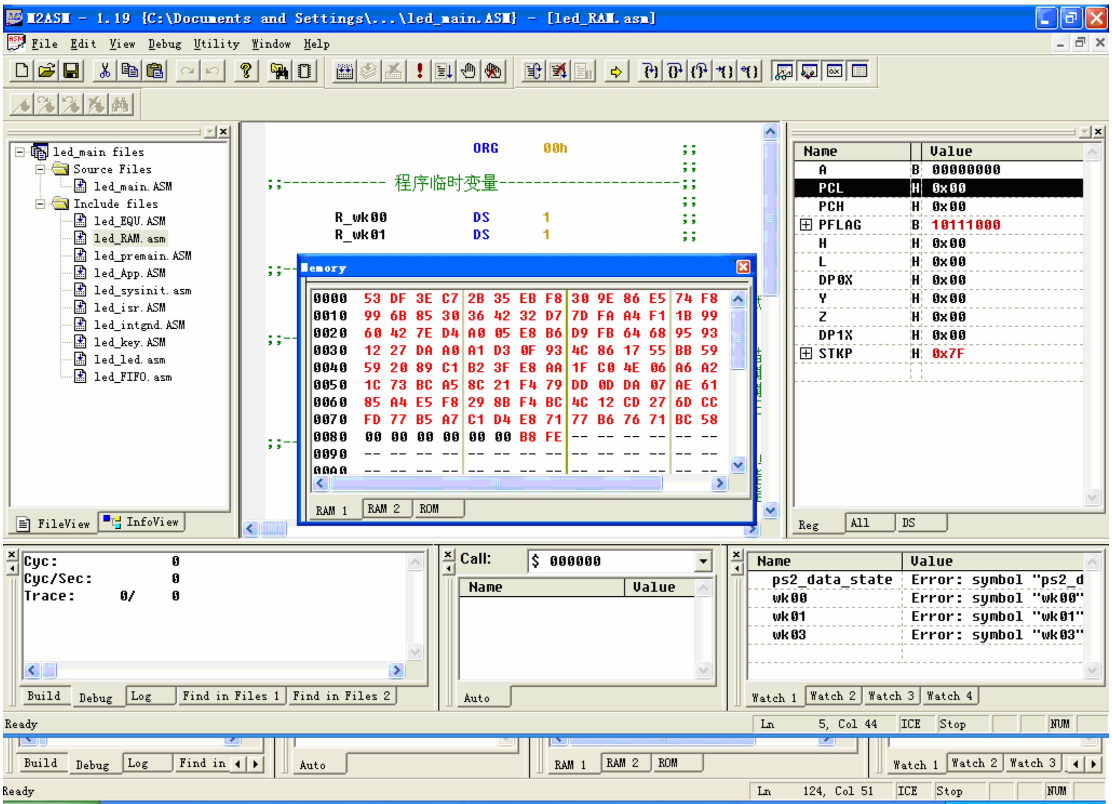



图 2-38 双击活动条弹出窗口

3. 工具栏或状态栏空白处单击右键，在弹出的菜单中可以通过选择 Output，Watch，Variables，Registers，

Memory，Workspace 等，选择是否显示相应的窗口。如图 2-39 所示：

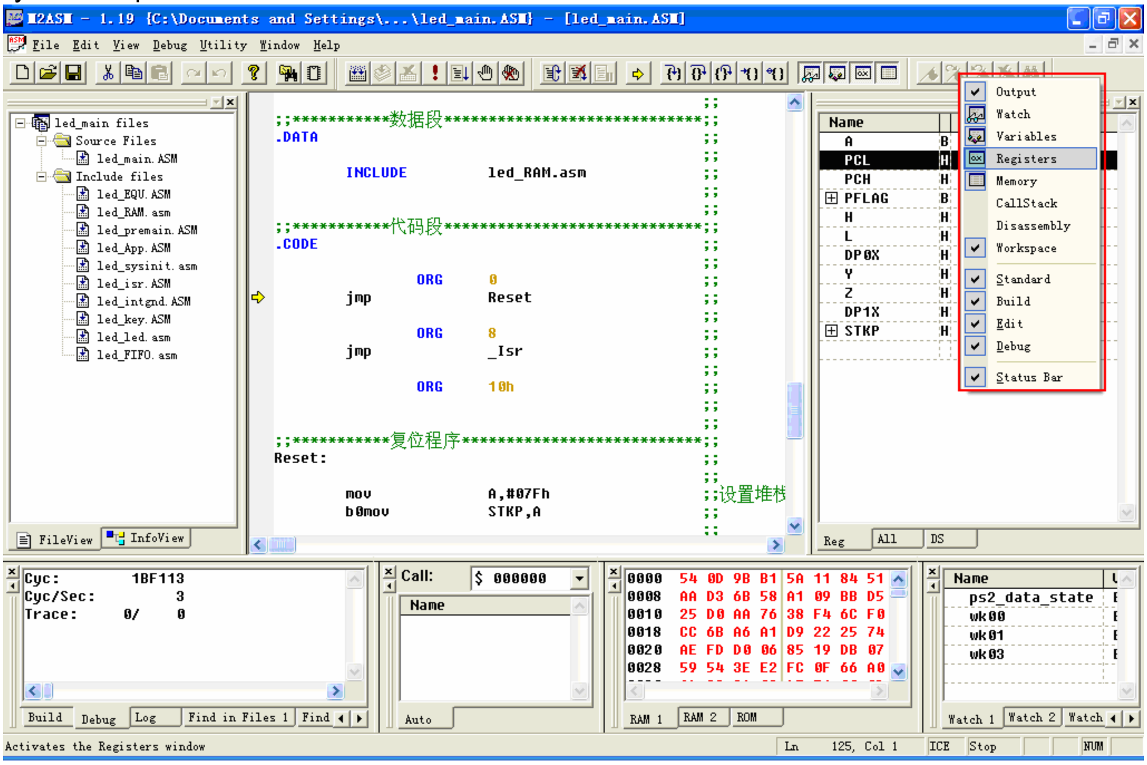


图 2-39 右击工具栏空白处选择显示或不显示窗口

如上图所示，名称前面的黑色的小对号或按下的图标表示该窗口已经打开。

# 2.3 创建和调试应用程序

## 2.3.1 创建工程/新建文件

M2IDE 集成开发环境下采用工程的方式来管理文件。所有的文件包括源程序（包括主程序和包含文件）、头文件以及说明性的文档，都可以放到同一个工程项目文件里统一管理。

创建一个应用程序的一般步骤如下：

- 新建一个工程项目文件；
- 创建一个源文件并输入程序代码；
- 保存创建的源码文件至项目文件夹中；
- 将源程序添加到项目中。

下面以创建一个新的工程文件 Test.PRJ 为例，具体说明如何建立一个应用程序。

双击桌面上的 M2Asm119 快捷图标，打开开发环境，如果是第一次使用 M2Asm，M2Asm 会自动打开 SN8Readme.txt，并显示欢迎对话框，如图 2-40 所示。选中对话框上的“Don't show SN8Readme.txt at next time”单击“OK”按钮，这样在下次打开 M2Asm 的时候不会再显示 SN8Readme.txt。如果不是第一次使用开发环境，M2Asm 启动的同时将打开前一次正确处理的工程。

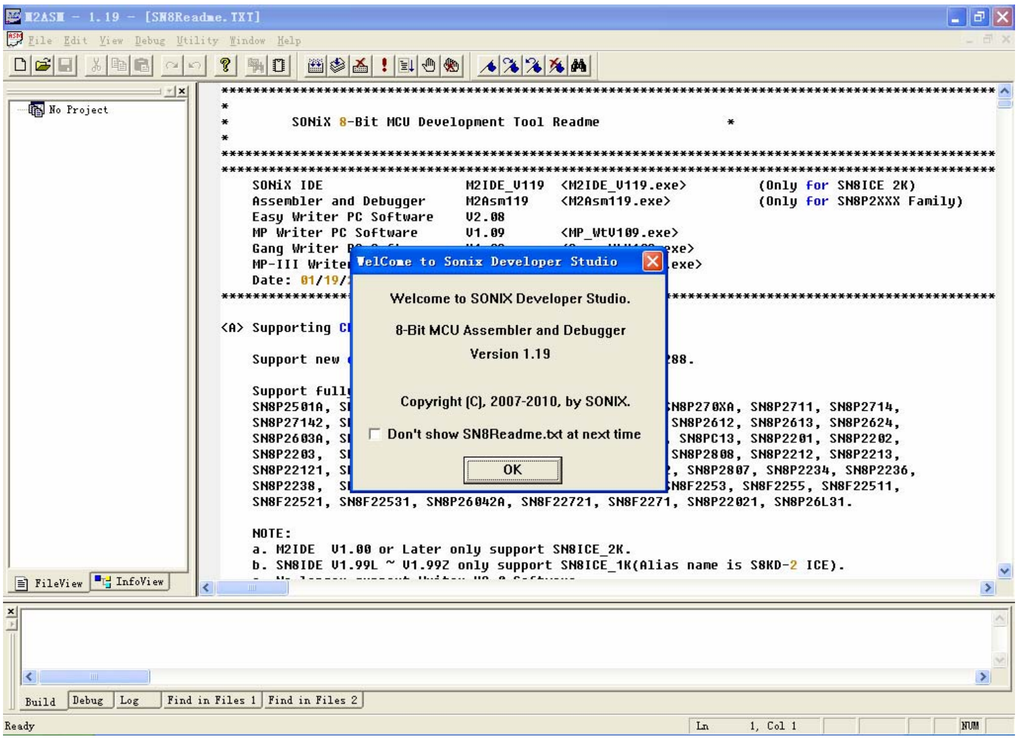



图 2-40 首次打开 M2Asm 界面

关闭所有文件和工程，M2Asm 的界面如图 2-41 所示，可见工程窗口、编辑区和编译消息栏均为空，工具栏上很多按钮均为灰色锁定状态。

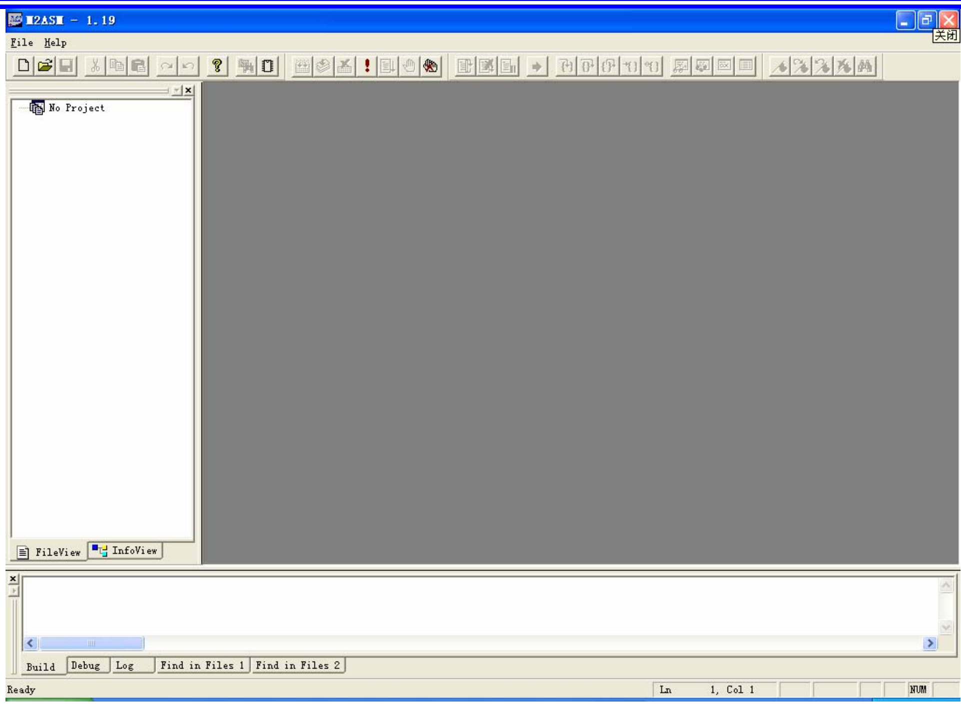



图 2-41 M2Asm 没有工程和文件打开状态下的界面

如果要建立工程，首先要建立一个添加到工程的文件，可以单击工具栏中的“New”按钮，也可使用菜单栏的“File”项，弹出如图 2-42 所示的菜单，选择“New”项。这样就可以成功创建了一个空白文档，文档的名称默认为 SONIX1，多次使用“New”会建立多个文件，文件的名称依次为 SONIX2、SONIX3 等 。

使用菜单栏的“File”项，弹出的菜单如图 2-43，观察可以发现这时的菜单与空闲时的有所不同，选择“Save”项，弹出如图 2-44 所示的文件保存对话框。

这里需要完成以下内容：

- 为文件重新定义一个名称，文件名称尽可能体现文件所能完成的功能；
- 选择文件存放的路径，建议文件和工程存放到相同的目录下，该工程的文件也都存放到该目录下，以方便管理；此处存放的路径为：D:\实验板程序\Test，文件名称为 Test，单击“保存”返回。

这时可以发现标题栏上的文件名称已经更改为 Test.ASM。

- 再次使用图 2-42 中的“File”菜单，选择“New Project”项，弹出如图 2-45 所示的文件打开对话框，选择刚刚建立的 Test 文件，单击打开，这样就成功建立一个工程。工程名与刚刚打开的文件名相同，均为 Test，同时 Test.ASM 文件被自动包含到了工程文件之中。标题栏中同时显示了当前打开工程的路径和当前文件的名称。

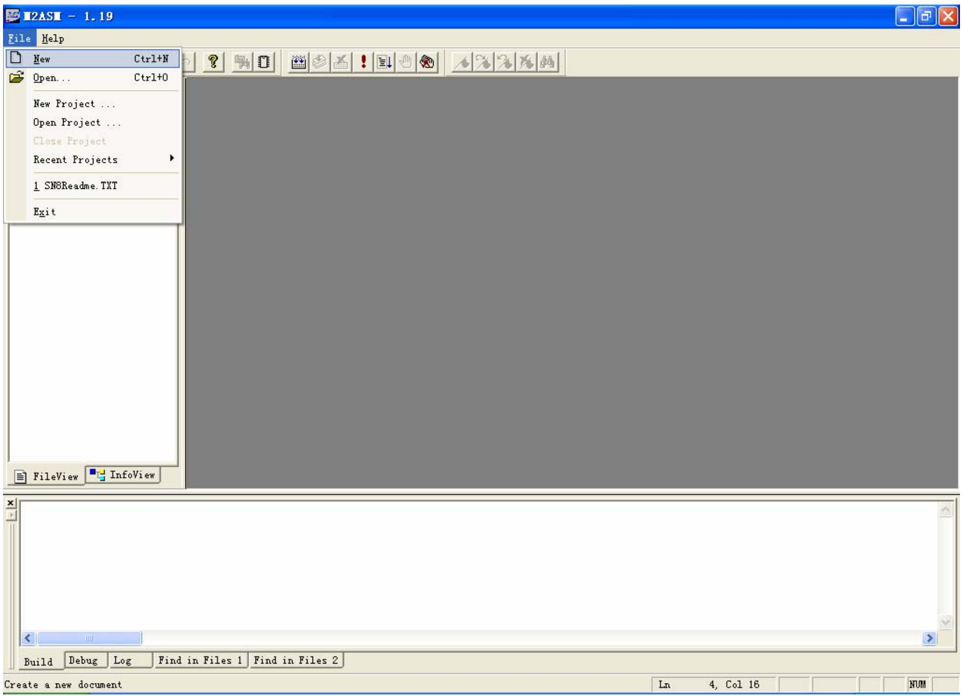



图 2-42 空闲状态下 File(文件)下拉菜单

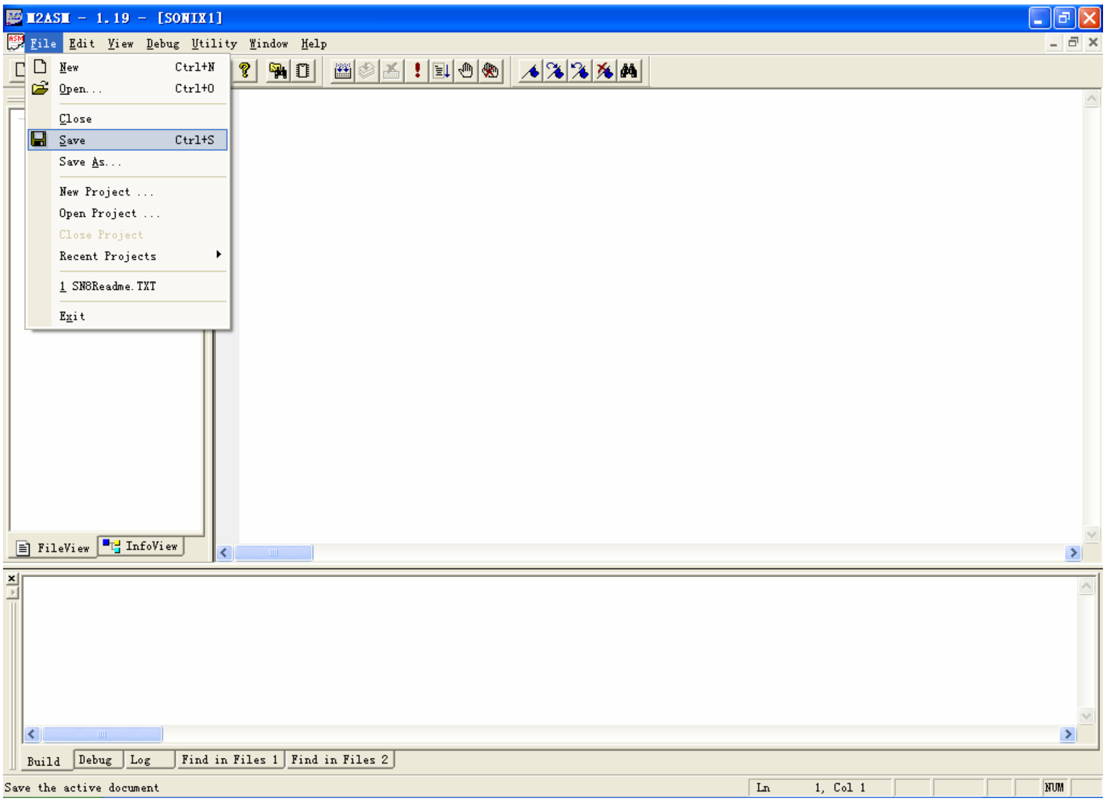



图 2-43 新建文件后 File(文件)下拉菜单

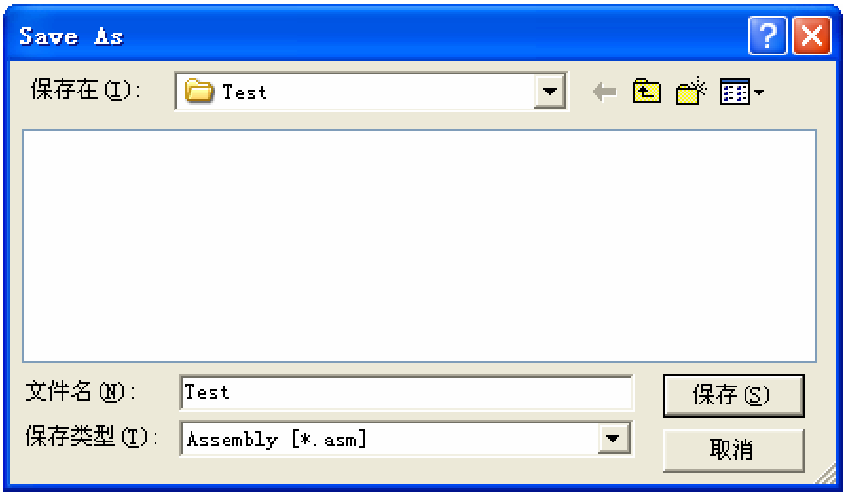



图 2-44 保存文件对话框

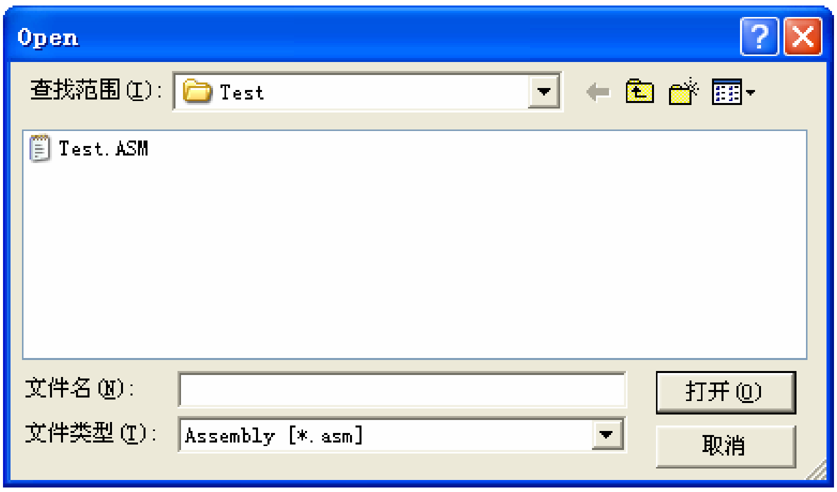



图 2-45 新建工程对话框

至此我们已经成功建立工程 Test.PRJ，并且在工程中包含了文件 Test.ASM，但是文件中没有任何的源程序，下面向该文件中输入自己的源程序 Test.ASM。M2Asm 中的文本输入方法同其它的文本编辑器基本相同，可以执行输入、删除、选择、拷贝和粘贴等基本的文字处理命令，不同的是汇编文件的编辑框内不同的内容被用不同的颜色来显示（如伪指令使用蓝色，注释使用绿色）。

需要注意的是在程序的开始需要使用“CHIP” 伪指令定义所用芯片的型号，这样在编译的过程中编译器会弹出不同的配置对话框，该内容将在文件的编译连接和调试一节中详细介绍。

实际情况中，一个工程往往包含多个程序文件，可以使用“INCLUDE”伪指令将他们包含到项目中来，在下次编译的过程中被包含的文件自动被加入工程中，需要注意的是包含文件应该尽可能和工程放在同一个目录中，否则在使用的“INCLUDE”时还需要指定被包含文件的路径。

如果工程中还需要添加头文件，可以在文件浏览窗口中的“Header Files”文件夹上右击，弹出如图 2-46 所示的菜单，选择“Add Files to Folder”后出现文件选择对话框，如图 2-47，选择要添加的头文件，单击“打开”完成头文件的添加。

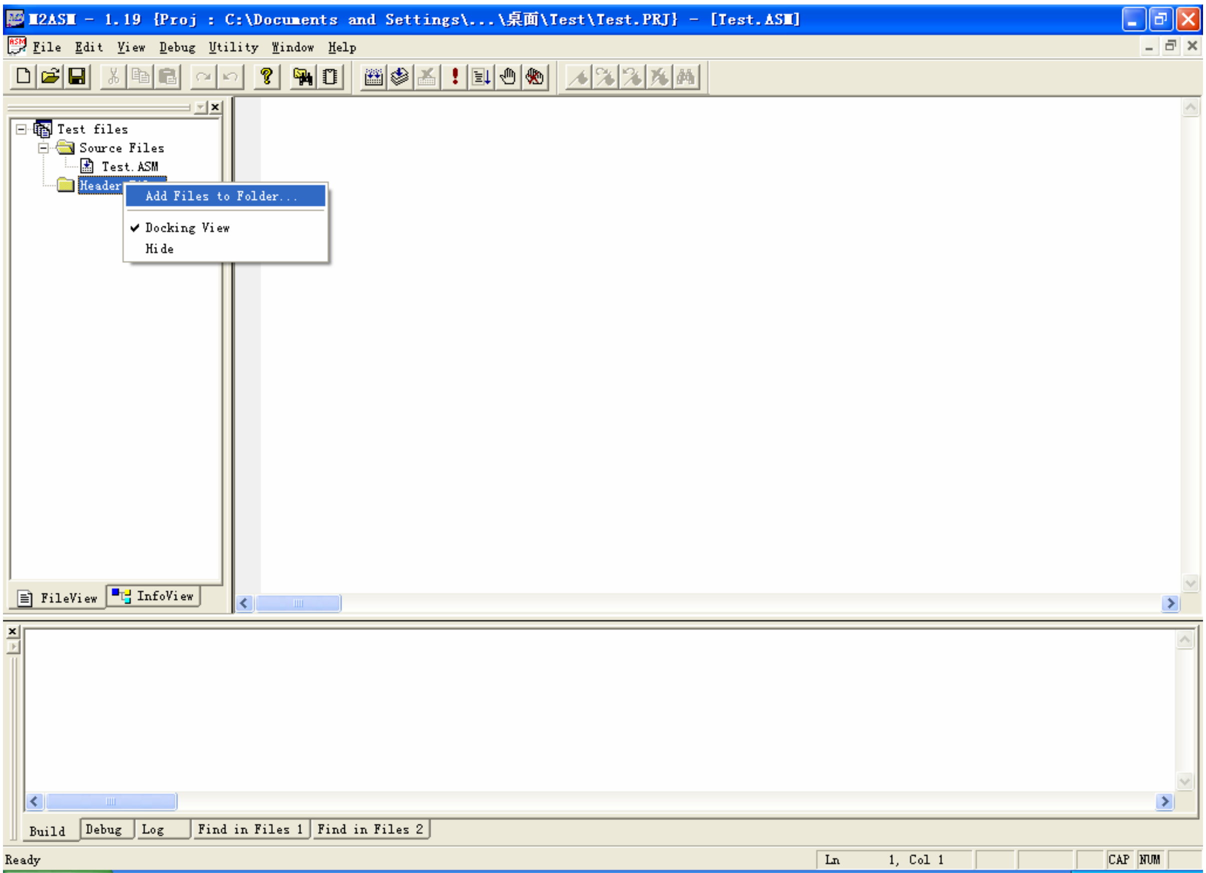



图 2-46 头文件添加菜单

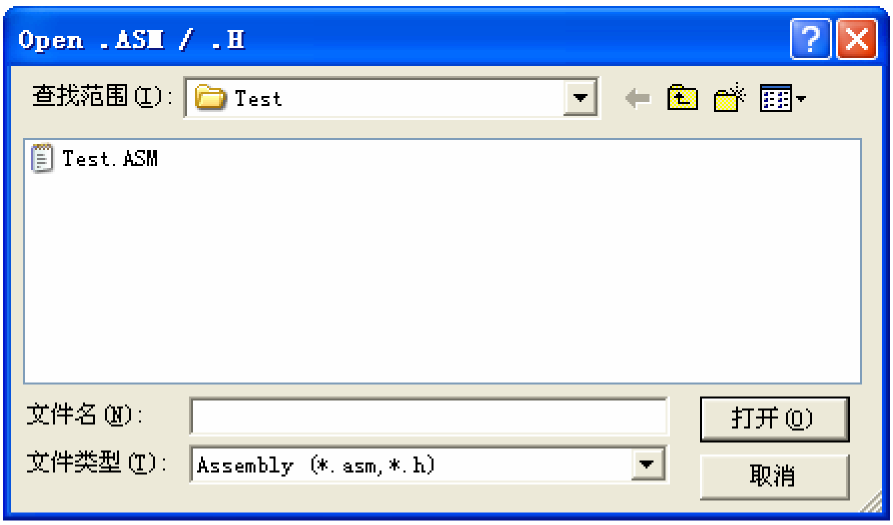



图 2-47 头文件选择对话框

下面讲述如何使用“INCLUDE”指定被包含文件的路径。

使用“INCLUDE”包含文件时，若文件放在与工程同一目录下时，其格式为：

```
INCLUDE        文件名称
```

举例如下：

```
INCLUDE     led_premain.ASM
INCLUDE     led_App.ASM
INCLUDE     SN88X.H            // 包含自定义的变量名
```

若文件不放在与工程同一目录下时，其格式为：

```
INCLUDE     文件路径和文件名称
```

举例如下：

```
INCLUDE    USB_ISP\SN8_USB_ISP_EnC.lib
INCLUDE    USB\SN8_USB_macro.h
INCLUDE    sub \ Filename.ASM
INCLUDE     C: \PROJECT.H        // 包含自定义的宏
INCLUDE     ..\Parent\File2.ASM
```

其中 USB_ISP ，USB 和 sub 分别为该工程目录下的文件夹，SN8_USB_ISP_EnC.lib ，SN8_USB_macro.h 和 Filename.ASM 文件包含在里面。

### 2.3.2 程序的编译和链接

通过上一节中建立工程并添加代码，下面我们可以开始对程序进行编译、链接了。使用 M2ASM 可以方便的对程序进行编译，一般分以下几个步骤。

使用工具栏中的“Build”按钮或单击“Debug”选择“Build”，也可以直接使用快捷键 F7 开始对工程进行编译、连接，此时弹出编译配置对话框，如图所示。




图 2-48 编译环境设置对话框

M2ASM是通过程序中使用“CHIP”伪指令来定义的芯片类型，并通过弹出不同的配置对话框对芯片相关属性进行配置的。图 2-48 中的为SN8P2708A系列芯片的配置对话框，但是各个选项均为灰色锁定状态，通过单击“Updata Code Option”可以激活各个选项，如图 2-49 所示，这时我们可以修改各个配置选项，各选项的功能说明请参考 **2.3.4 编译选项（Code Option选项）**章节。

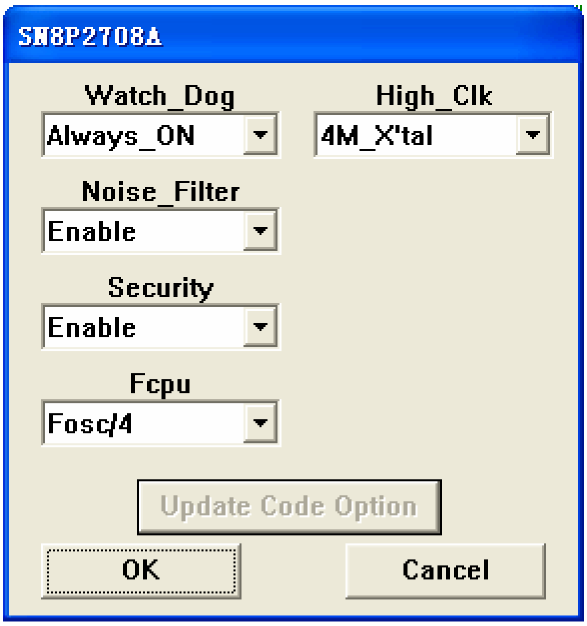



图 2-49 激活状态下的设置对话框

配置完毕后单击“OK”完成编译配置，如果程序没有错误的话，代码中定义芯片类型语句之后出现图 2-50 所示的配置信息。

```
CHIP            SN8P2708a
//{{SONIX_CODE_OPTION
        .Code_Option    Noise_Filter    Enable
        .Code_Option    Watch_Dog       Always_ON       ; Watchdog still enable even in Green and Sleep mode
        .Code_Option    High_Clk        4M_X'tal        ; Crystal/Resonator: 2Mhz~10Mhz
        .Code_Option    Fcpu            #2      ; Fcpu = Fosc/4
        .Code_Option    Security        Enable

//}}SONIX_CODE_OPTION
```

图 2-50 编译配置信息

在实际情况中能够一次性通过编译是很难做到的，代码中时常会出现一些错误而不能通过编译，如当执行编译指令后 M2ASM 弹出如图 2-51 下错误信息提示框，其中显示了出错的行、警告个数和错误个数。

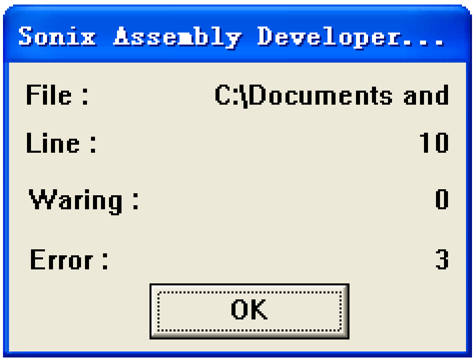



图 2-51 编译错误信息提示框

单击“OK”将错误信息框消除，此时界面如图 2-52 所示，可以发现在编辑框中出错的语句前被用一个蓝色箭头标记。在编译信息窗口中，编译错误的信息被罗列出来，包含了错误的类型和位置，蓝色反白显示的信息为当前编辑框中所指错误的错误信息。用户可以通过双击错误信息的方法来显示当前的错误行，例如我们双击第一条错误信息时，第一个错误信息被蓝色反白显示，编辑框中蓝色箭头指向第一个错误语句。

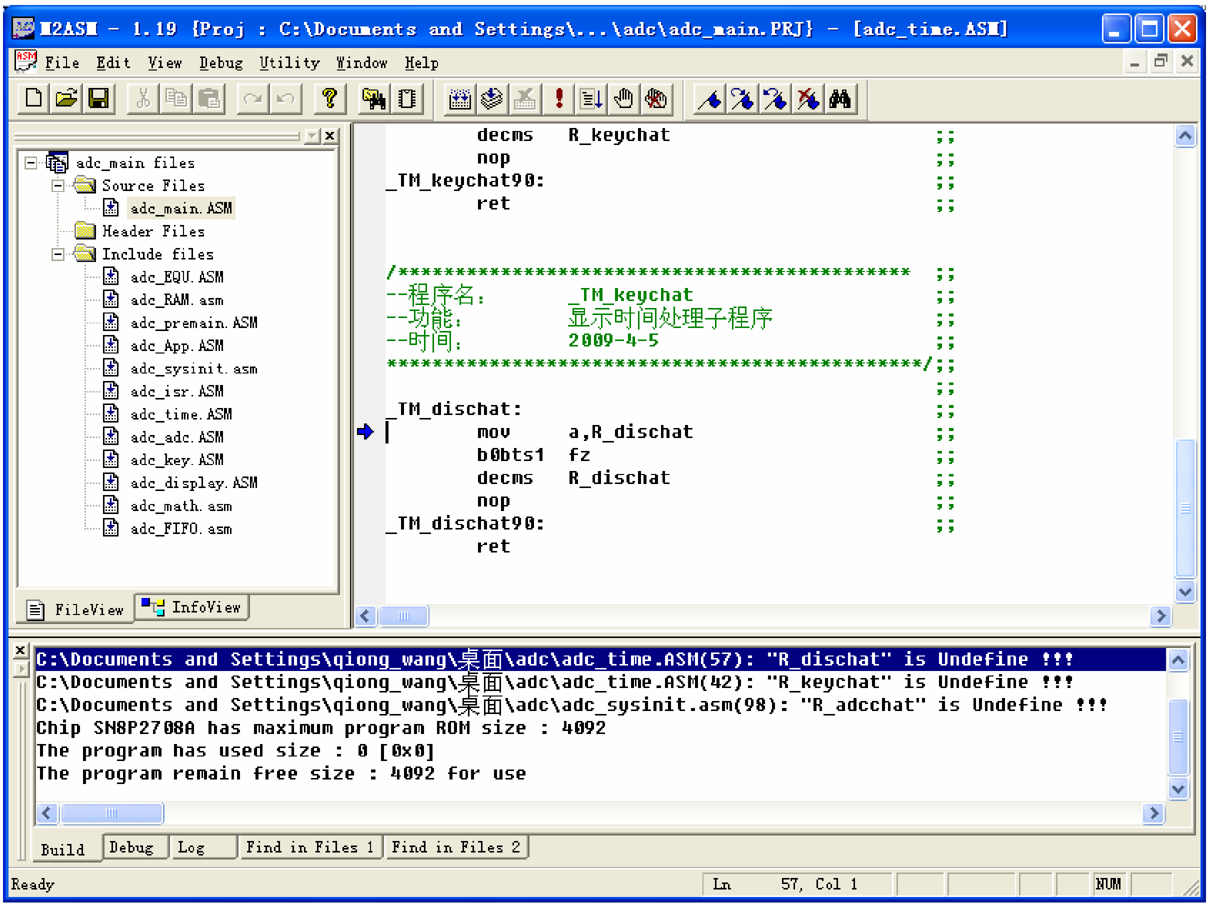



图 2-52 编译出错界面

修改错误后，重新编译，成功编译后编译信息框中的内容如图 2-53，其中显示了校验和、所选芯片的程序存储空间、代码占用空间和剩余空间。

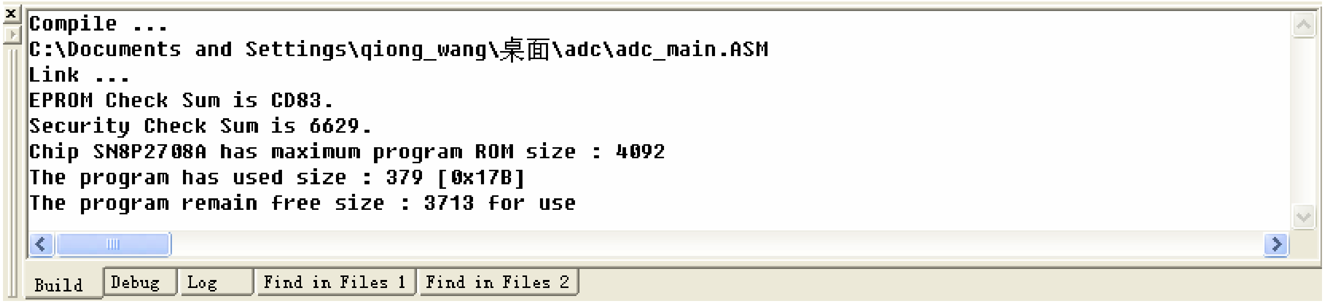



图 2-53 成功编译后的编译信息框

## 2.3.3 程序的运行与调试

为了用户更加方便的分析和调试程序，M2ASM 集成开发环境提供了很多调试命令和观察窗口，在调试过程中，善于使用开发环境中的各种窗口，对程序运行的状态、变量的变化情况等的观察，可以做到事半功倍的效果。下面我首先介绍常用的调试命令，然后用一个实例来说明调试过程。应该注意调试命令和调试窗口只有在调试状态下才是有效的。

### 调试命令

在 M2ASM 环境下有 3 种方法来执行 M2ASM 提供的调试命令。

方法一：在调试状态下，使用 M2ASM 菜单栏的调试工具菜单（Debug），弹出调试命令菜单，选择相应的调试命令即可执行该命令；

方法二：在调试状态下，单击 M2ASM 工具栏中的快捷命令图标，如图 2-55 所示，也可以执行相应的命令。

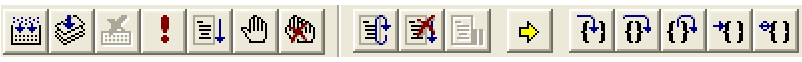


图 2-54 快捷命令图标

方法三：使用快捷键，快捷键的分布见表 2-2。熟练的使用快捷键可以大大提高程序调试速度。

表 2-2　调试命令快捷键

| 快捷键 | 命令 |
|---|---|
| F5 | 启动/停止运行 |
| Ctrl+ F5 | 复位 |
| Shift+ F5 | 退出调试状态 |
| F7 | 编译程序 |
| F8 | 下载程序至仿真器 |
| F9 | 设置断点 |
| Ctrl+ Shift+F9 | 清除所有断点 |
| F10 | 跳过函数 |
| Ctrl+F10 | 运行到光标处 |
| F11 | 单步运行 |
| Shift+F11 | 跳出函数 |
| F12 | 程序计数器指向光标所在处 |

在集成开发环境下，系统为用户提供了多种运行程序的方法，用户可以通过相应的命令来实现全速、单步等多种方法运行程序。

常用的运行方式有以下几种：

**全速运行/暂停运行（F5）**

在停止或暂停状态下，执行此命令将全速运行用户的应用程序，在这种方式下，可以查看程序的功能实现情况。实际中，该命令一般和断点一起使用，若已经在程序的关键部分设置了断点，执行该命令后程序将执行到断点处，且运行指针指向该程序行，并等待执行其它命令。在运行过程中，执行该命令程序将停止在当前指令行。

**单步运行（F11）**

此命令执行当前光标所指向的命令语句，执行之后 PC 指向下一条指令。如果执行的行为调用的子程序（或称函数），则会跳入子程序内部。该命令方便用户精确的查看每条指令的执行情况，结合各观察窗口，用户也可以查看程序的执行对寄存器的影响。

**跳过函数（F10）**

该命令的功能也是执行当前行指令，与单步运行不同的是，如果执行的语句为函数调用语句，该命令将一次执行完该函数，而不进入函数的内部。如果执行的语句为一般汇编语句，功能与单步运行相同。

**跳出函数（Shift+F11）**

此命令用于跳出当前所在的子程序，如果希望快速的执行完当前所处的子程序，并回到函数被调用的位置时，可以使用该命令。需要特别注意的是，如果当前没有处在任何子程序中，请尽量不要使用该命令，否则可能会导致仿真的失败。

**运行到光标处（Ctrl+F10）**

执行此命令可使程序执行到代码窗口中光标所在的位置。这相当于把光标所在的位置作为一个临时的断点。

**程序计数器指向光标（F12）**

执行此命令可以使程序计数器指向当前光标所在的行，注意这里只修改程序计数器 PC，并不执行任何的指令。

**复位（Ctrl+ F5）**

执行该指令，程序计数器被置 0，需要注意，这个命令不能使外部设备和系统寄存器进入复位状态，因此这个复位命令并不等同于 CPU 的硬件复位。

**下载程序（F8）**

SN8ICE 2K 支持程序的下载，下载后程序可以脱离计算机环境自由运行。使用该命令后将弹出如图 2-55 所示的文件选择对话框，选择要下载的.SN8 文件，单击打开，弹出图 2-56 所示的下载成功提示框，提示用户下载文件成功，MCU 正在自由运行。

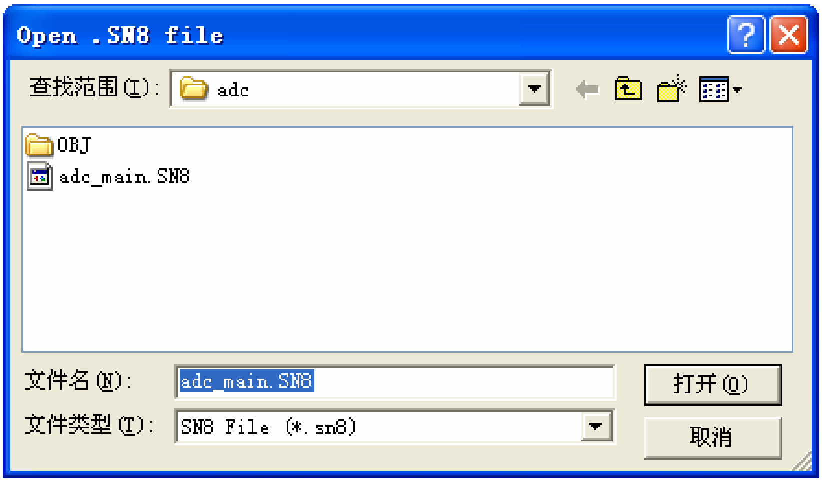



图 2-55 下载文件选择对话框




图 2-56 下载成功提示框

以下说明 M2ASM 调试程序的过程。

在编译成功的条件下，执行 Go（运行）命令，进入如图 2-57 所示的调试环境，可以看到开发环境的工具栏出现了一些之前没有显示的调试命令按钮，并且各个观察窗口也显示出来。此时程序的指针指向第一条跳转指令“jmp reset”，程序也将从此处开始执行。这里有两点需要特别说明：

M2ASM 没有提供软件模拟仿真的功能，故在进入调试环境之前必须要将仿真器连接到计算机上，否则将不能进入仿真环境，并出现如图 2-58 所示的提示框；

如果在没有编译之前直接使用 Go（运行）命令，M2ASM 将首先执行编译动作，然后直接进入调试状态。

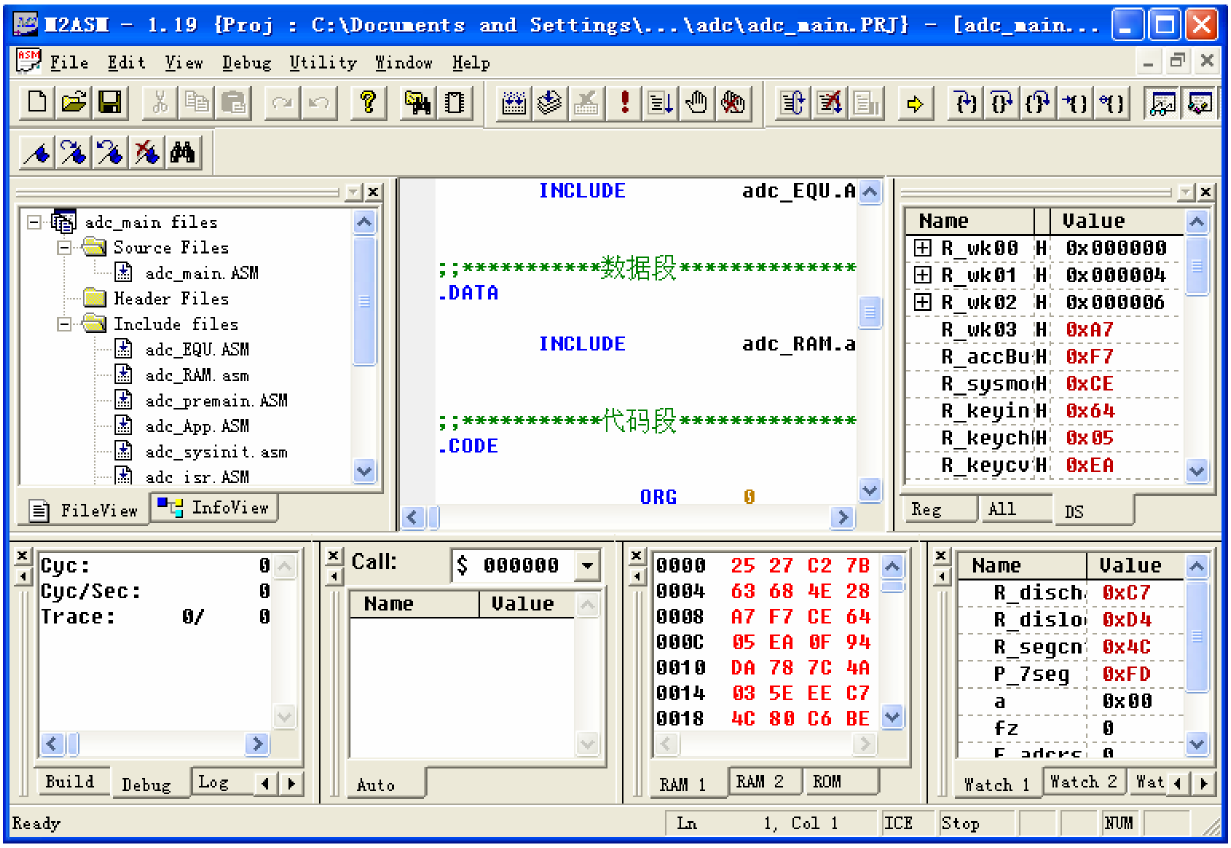



图 2-57 调试界面




图 2-58 未找到仿真器提示框

连续执行 Step Into（单步运行）指令，可以看到当前变量观察窗口中的变量不断的变化，并显示出了各自的值，当寄存器或用户定义变量发生改变时，寄存器窗口中寄存器或变量值变成红色显示。寄存器窗口中的数组在程序运行前只显示该数组所占用的地址单元，数组成员的具体信息需要单击该数组前的“+”号即可展开该数组，如图 2-59 所示，可以看出数组 rled_buf 共有 4 各成员，它们的值均为 0x17。

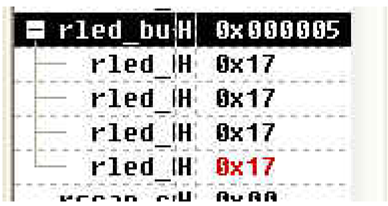



图 2-59 观察窗中展开的数组

将光标移动到子程序 mnled 的开始处，点击图标，此时在 mnled 的第一条指令前面出现一红色的实心圆点，

表示在本指令处设置了一个断点。

执行 Go（运行）命令，程序直接运行到了所设置的断点处，如图 2-60 所示。在观察窗口中可以发现在刚才的程序段中被改变的值呈红色显示，在观察窗口中输入两个变量的名称，分别为：“rled_buf”和“r500ms_cnt”，可以直接观察到它们当前的数值，如图 2-61 所示。

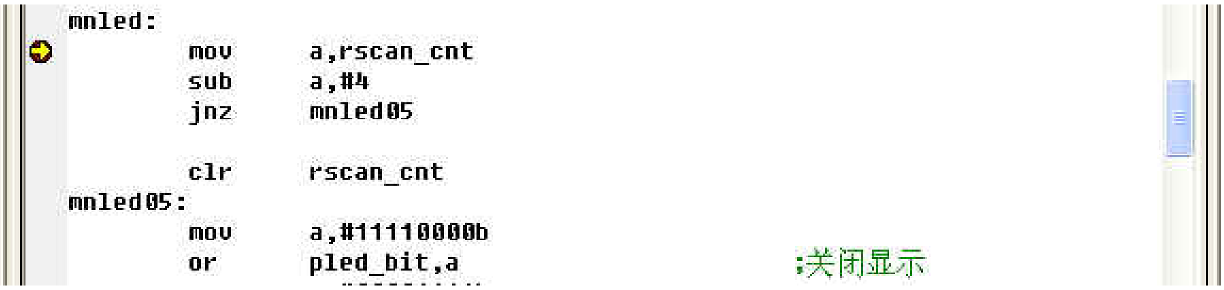



图 2-60 程序运行到断点

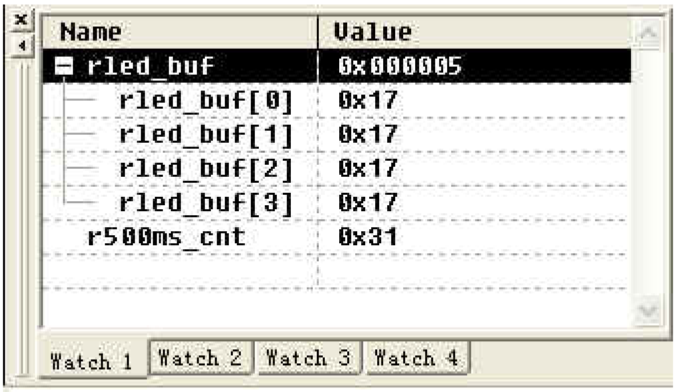



图 2-61 在变量观察窗口直接观察变量的值

执行 Step Out（跳出函数）命令，此时发现程序直接执行完子程序并停止在调用指令的下一条语句处，如图 2-62 所示。

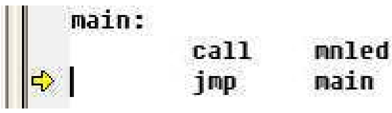



图 2-62 跳出函数

单击图标，取消前面设置的断点，在调用显示子程序语句处设置断点，并运行到该条指令处，如图 2-64 所示。执行 Step Over（跳过函数）命令，可以发现程序没有进入 mnled 子程序，而是直接运行完该子程序，停止在调用指令的下一条指令处，如图 2-64 所示。

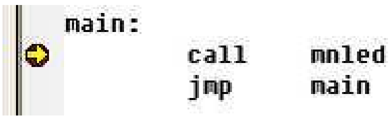



图 2-63 运行到调用显示函数语句处




图 2-64 跳过函数

执行 Remove All Breakpoints（清除所有断点）命令，可以看到所有的断点被清除掉了。

再次执行 Go（运行）命令，使程序全速运行，可以观察到界面如图 2-65 所示，可以再次执行 Go（运行）命令来暂停程序的运行。

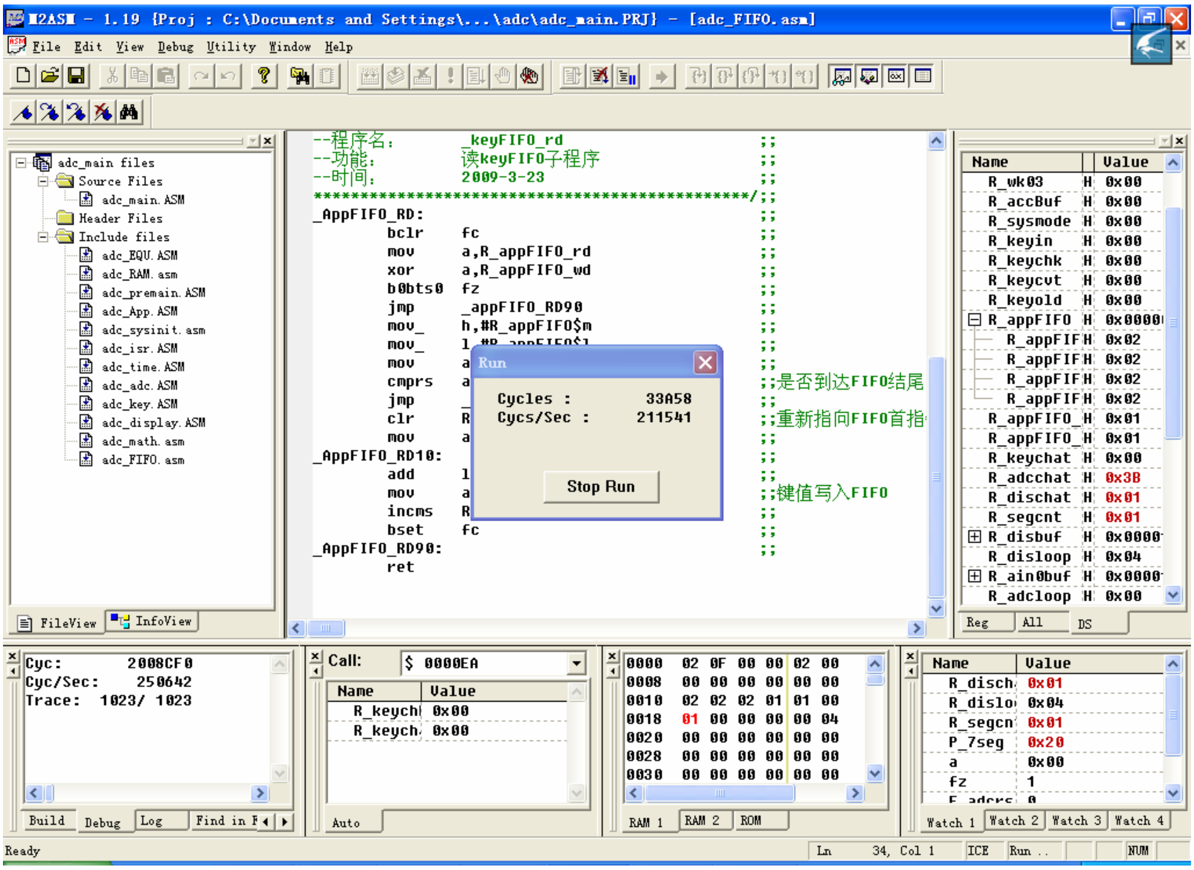


图 2-65 全速运行界面

停止程序运行，使用 Download（下载）命令将程序下载到仿真器中，此时关闭 M2ASM 开发环境，程序已经在仿真器中自由运行了。

以上详细的介绍了在 M2ASM 集成开发环境下调试程序的常用方法，用户在使用时可以灵活的运用开发环境提供的各种工具和调试命令，只有这样才能更快更好的编写出高质量的程序。

## 2.3.4 编译选项（Code Option选项）

在 2.3.2 章节 程序的编译和链接中已经讲到编译选项 Code Option。如图 2-66 所示。

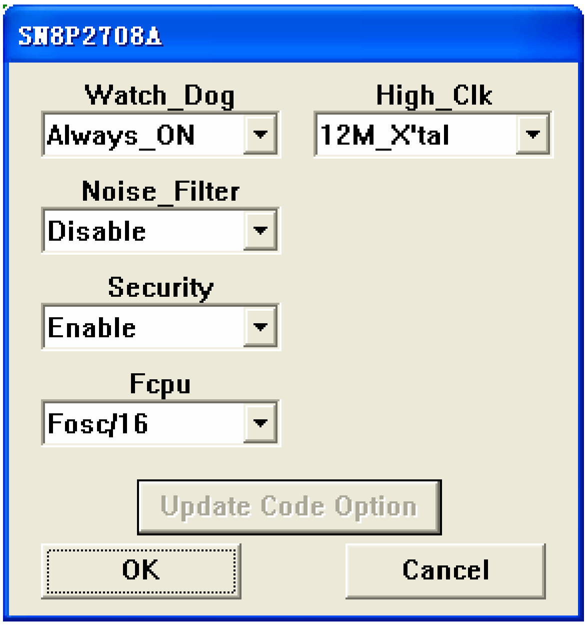



图 2-66 编译选项

不同的 chip 宣告和不同版本的编译软件，Code Option 有不同的选项可供选择。部分选项含义如下：

◆ Watchdog：
Always on－看门狗定时器一直开启；
Enable－看门狗定时器在 normal 和 slow 模式下开启，在 green 和 sleep 模式下停止；
Disable－看门狗定时器关闭。
※ 注：当选择 Always on 选项时，系统将无法进入 sleep 模式。

◆ Reset_Pin：
Pxx－选择内部复位，同时该引脚将作为单向输入口 Pxx 使用；
Reset－选择外部复位；
※ 注：当选择内部复位时，Pxx 口为单向输入口，且无内部上拉电阻。

◆ High_Clk：
IHRC_16M－芯片工作振荡源采用内部 16M 高速 RC 振荡电路；
Ext_RC－芯片工作振荡源采用外部 RC 振荡电路；
32K_X'tal－芯片工作振荡源采用外部低频率晶振（例如 32.768KHz）；
4M_X'tal－芯片工作振荡源采用外部标准石英或陶瓷振荡器（一般在 2M ～ 10MHz）；
12M_X'tal－芯片工作振荡源采用外部高速石英或陶瓷振荡器（一般在 10MHz ～ 16MHz）。
※ 注：IHRC_16M 选项只有在内部集成了高速 RC 振荡电路的 IC 型号中才会出现，当选择此项时，XIN/XOUT 两个引脚将作为一般 I/O 使用。

◆ Fcpu：
Fosc/1－指令周期 = 1 个时钟周期；
Fosc/2－指令周期 = 2 个时钟周期；
Fosc/4－指令周期 = 4 个时钟周期；
Fosc/8－指令周期 = 8 个时钟周期；
Fosc/16－指令周期 = 16 个时钟周期；
※ 注：当在 Code Option 中选择 Noise_Filter Enable 或 IHRC_16M 时，Fcpu 选项里的 Fosc/1 和 Fosc/2 两项将被自动屏蔽。

◆ Security：
enable－程序代码加密；
disable－程序代码不加密。

◆ Noise_Filter：
enable－打开噪声滤波功能。
disable－关闭噪声滤波功能。
※ 注：当开启噪声滤波功能后，会提高芯片的抗干扰能力，同时 Fcpu 选项里的 Fosc/1 和 Fosc/2 两项将被自动屏蔽。

◆ LVD：
LVD_L－VDD 低于 2.0V 时，LVD 复位系统；
LVD_M－VDD 低于 2.0V 时，LVD 复位系统，LVD 的 24-bit PFLAG 寄存器作为 2.4V 低电压监测器；
LVD_H－VDD 低于 2.4V 时，LVD 复位系统，LVD 的 36-bit PFLAG 寄存器作为 3.6V 低电压监测器；
LVD_MAX－VDD 低于 3.6V 时，LVD 复位系统。（个别型号具有，如 SN8P2522）

◆Rst_Length
No－没有外部复位消抖时间；
128*ILRC－外部复位消抖时间为 128*ILRC。

◆Ext_OSC
6MHz—芯片工作振荡源选择 6MHz 晶振/陶瓷振荡器；
12MHz－芯片工作振荡源选择 12MHz 晶振/陶瓷振荡器；
16MHz－芯片工作振荡源选择 16MHz 晶振/陶瓷振荡器。

◆Fslow
Flosc/2－低速模式时钟为 Flosc/2；
Flosc/4－低速模式时钟为 Flosc/4。

◆Ext_Reset_Length
No－没有外部复位消抖延时时间；
128*ILRC－外部复位消抖延时时间为 128*ILRC。

◆IHRC_Detect
Enable－开启 IHRC 检测功能；
Disable—不开启 IHRC 检测功能。

## 2.3.5 工程文件类型

| | |
|---|---|
| .ASM | 汇编语言程序文件 |
| .HEX | 可用于烧录的十六进制文件 |
| .LST | 列表文件 |
| .PRJ | 工程项目，.PRJ 为名称的后缀名 |
| .SN8 | 系统编译后生成的烧写档 |
| .LIB | 库文件 |
| .BBB | 使用联机模式 Read OTP 后生成的文档 |
| .INI | 记录 RollingCode 内容的文件 |

## 2.4 如何仿真LCD

LCD Simulator Function 是在 MCU（内嵌 LCD 驱动）仿真时，为方便开发人员调试程序及简化硬件电路，利用软件接口模拟 LCD 显示的功能。不需要 LCD 屏，只需要根据 LCD 屏的开模方式编辑出相应的 BMP，在软件仿真的时候加载 BMP 图形编辑后生成的.LCD 文件即可。

LCD 仿真面板如图 2-67 所示。

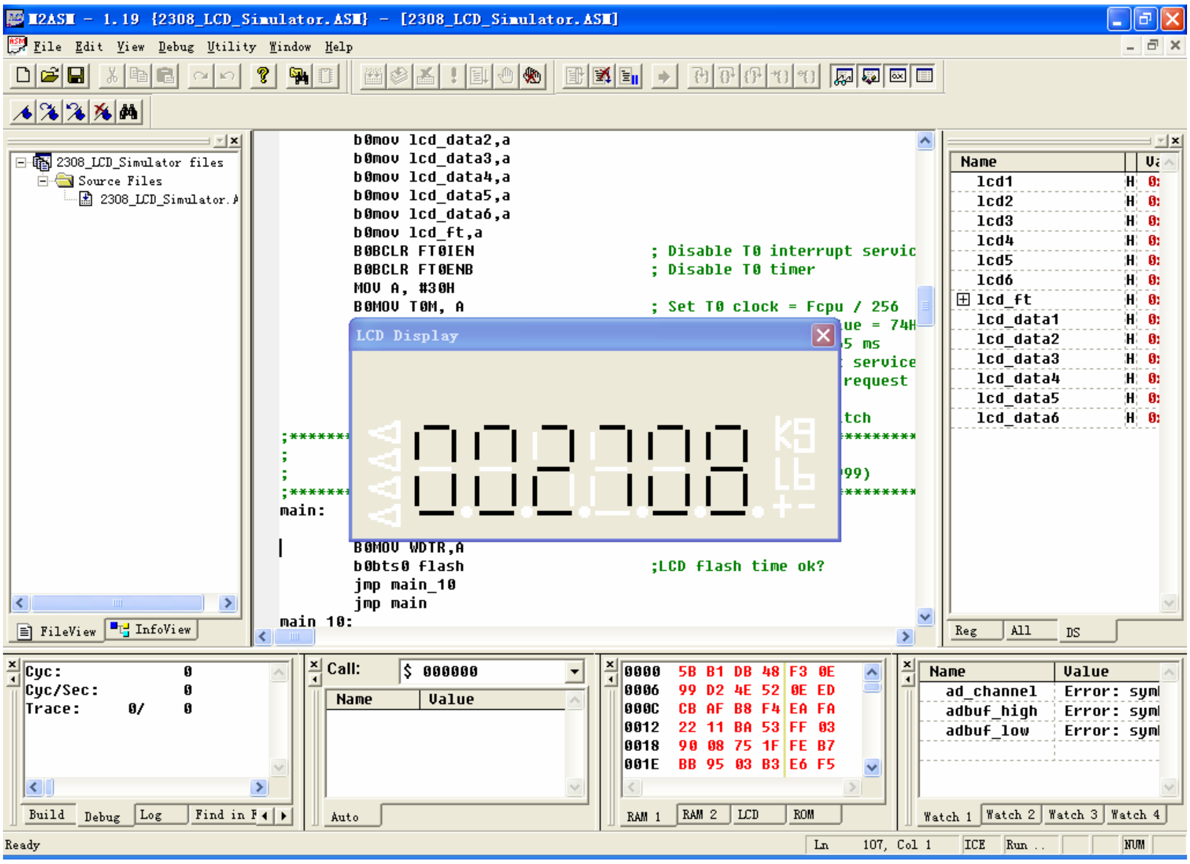



图 2-67 LCD 软件仿真面板

关于 LCD 的具体仿真方法可以参考编译器安装目录下 LCD_Simulator_Manual_V01_SC.pdf 文件；
路径：C:\Sonix\M2IDE_Vxxx\LCD_Simulator_Manual_V01_SC.pdf

# 第 3 章 开发语言

## 3.1 指令集

| Field | 指令格式 | 描述 | C | DC | Z | 周期 |
|---|---|---|---|---|---|---|
| | MOV A,M | A ← M | - | - | √ | 1 |
| M | MOV M,A | M ← A | - | - | - | 1 |
| O | B0MOV A,M | A ← M (bnak 0) | - | - | √ | 1 |
| V | B0MOV M,A | M (bank 0) ← A | - | - | - | 1 |
| E | MOV A,I | A ← I | - | - | - | 1 |
| | B0MOV M,I | M← I。 (M 仅适用于系统寄存器R, Y, Z , RBANK, PFLAG) | - | - | - | 1 |
| | XCH A,M | A ←→M | - | - | - | 1+N |
| | B0XCH A,M | A←→M (bank 0) | - | - | - | 1+N |
| | MOVC | R, A← ROM [Y,Z] | - | - | - | 2 |
| | ADC A,M | A ← A + M + C，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1 |
| A | ADC M,A | M ← A + M + C，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1+N |
| R | ADD A,M | A ← A + M，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1 |
| I | ADD M,A | M ← A + M ，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1+N |
| T | B0ADD M,A | M (bank 0) ← A + M (bank 0)，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1+N |
| H | ADD A,I | A← A + I，如果产生进位则C=1，否则C=0 | √ | √ | √ | 1 |
| M | SBC A,M | A ← A - M - /C，如果产生借位则C=0，否则C=1 | √ | √ | √ | 1 |
| E | SBC M,A | M ← A - M -/C，如果产生借位则C=0，否则C=1 | √ | √ | √ | 1+N |
| T | SUB A,M | A ← A – M，如果产生借位则C=0，否则C=1 | √ | √ | √ | 1 |
| I | SUB M,A | M ← A – M ，如果产生借位则C=0，否则C=1 | √ | √ | √ | 1+N |
| C | SUB A,I | A← A – I，如果产生借位则C=0，否则C=1 | √ | √ | √ | 1 |
| | DAA | 将ACC 中的数据由十六进制改为十进制格式 | √ | - | - | 1 |
| | MUL A,M | R, A ← A * M，乘积低字节存放到ACC，高字节存放在系统寄存器R 中，ZF 标志位受ACC 内容影响 | - | - | √ | 2 |
| | AND A,M | A ← A 与M | - | - | √ | 1 |
| L | AND M,A | M ← A 与M | - | - | √ | 1+N |
| O | AND A,I | A ← A 与I | - | - | √ | 1 |
| G | OR A,M | A ← A 或M | - | - | √ | 1 |
| I | OR M,A | M← A 或M | - | - | √ | 1+N |
| C | OR A,I | A← A 或I | - | - | √ | 1 |
| | XOR A,M | A ← A 异或M | - | - | √ | 1 |
| | XOR M,A | M ← A 异或M | - | - | √ | 1+N |
| | XOR A,I | A← A 异或I | - | - | √ | 1 |
| | SWAP M | A (b3~b0, b7~b4) ←→ M(b7~b4, b3~b0) | - | - | - | 1 |
| P | SWAPM M | M(b3~b0, b7~b4) ←→ M(b7~b4, b3~b0) | - | - | - | 1+N |
| R | RRC M | A← M 带进位右移 | √ | - | - | 1 |
| O | RRCM M | M← M 带进位右移 | √ | - | - | 1+N |
| C | RLC M | A ←M 带进位左移 | √ | - | - | 1 |
| E | RLCM M | M← M 带进位左移 | √ | - | - | 1+N |
| S | CLR M | M← 0 | - | - | - | 1 |
| S | BCLR M.b | M.b ← 0 | - | - | - | 1+N |
| | BSET M.b | M.b← 1 | - | - | - | 1+N |
| | B0BCLR M.b | M(bank 0).b ← 0 | - | - | - | 1+N |
| | B0BSET M.b | M(bank 0).b ← 1 | - | - | - | 1+N |
| B | CMPRS A,I | 比较，如果相等则跳过下一条指令C 与ZF 标志位可能受影响 | √ | - | √ | 1+S |
| | CMPRS A,M | 比较，如果相等则跳过下一条指令C 与ZF 标志位可能受影响 | √ | - | √ | 1+S |
| R | INCS M | A ← M + 1 ，如果A = 0，则跳过下一条指令 | - | - | - | 1+S |
| A | INCMS M | M ← M + 1 ，如果M = 0，则跳过下一条指令 | - | - | - | 1+N+S |
| N | DECS M | A ← M – 1，如果A = 0，则跳过下一条指令 | - | - | - | 1+S |
| C | DECMS M | M ← M – 1，如果M = 0，则跳过下一条指令 | - | - | - | 1+N+S |
| H | BTS0 M.b | 如果M.b = 0，则跳过下一条指令 | - | - | - | 1+S |
| | BTS1 M.b | 如果M.b = 1，则跳过下一条指令 | - | - | - | 1+S |
| | B0BTS0 M.b | 如果M(bank 0).b = 0 ，则跳过下一条指令 | - | - | - | 1+S |
| | B0BTS1 M.b | 如果M(bank 0).b = 1 ，则跳过下一条指令 | - | - | - | 1+S |
| | JMP d | 跳转指令，PC15/14 ← RomPages1/0，PC13~PC0← d | - | - | - | 2 |
| | CALL d | 子程序调用指令，Stack ← PC15~PC0，PC15/14← RomPages1/0，PC13~PC0 ← d | - | - | - | 2 |
| M | RET | 子程序跳出指令，PC← Stack | - | - | - | 2 |
| I | RETI | 中断处理程序跳出指令，PC ← Stack ，使能全局中断控制位 | - | - | - | 2 |
| S | RETLW | 子程序跳出指令， PC← Stack ，并将取得的值存放在ACC 中 | - | - | - | 2 |
| C | NOP | 空指令，无特别意义 | - | - | - | 1 |

**1 指令系统中的符号说明**

从上面几个章节及指令表可以看出，在操作数和目的操作数字段使用了一些符号。这些符号的含义归纳如下：

- A ：累积器 ACC。
- M ：数据存储器 RAM 中的一个单元地址。
- I ：常数，开头须有符号"＃"，例如：＃0x1A, #V1
- ．b ：位地址，例如：M.5 表数据存储器的 M 单元的第 5 位
- d ：程序存储器 ROM 的地址。
- (X：Y)：字节 X 和 Y 组成一个字 (word)，其中 Y 为低字节。
- (X：Y：Z)：由 X、Y 和 Z 三个 8 位数组一个 24 位的数据。

执行周期中

N：系统寄存器为 1，用户自定义的寄存器为 0；

S：条件为真则为 1，条件不为真则为 0；

**2 指令对标志位的影响**

SONiXMCU 指令按对标志位的影响情况分为两类：一类指令执行后要影响到程序状态寄存器 PFLAG 的某些标志位的状态，即不论指令执行前标志位状态如何，指令执行时总按标志位的定义形成新的标志状态；另一类指令执行后不会影响到标志位的状态，原来是什么状态，指令执行后还是这个状态。

不同的指令对标志位的影响是不相同的，每条指令对标志位的影响如指令表所示。

注：由于 **SONiX** 的 **SN8P1000** 系列与 **SN8P2000** 系列 **MCU** 的部分指令周期不同，可能跟此表描述有所出入，而且部分 **MCU** 的实际指令可能只有列表中的部分而并非全部，因此以 **MCU** 的规格书描述为准，此表仅仅用来列出并介绍各条指令的用法。

## 3.2 伪指令

伪指令是用于告诉汇编程序如何进行汇编的指令，它既不控制机器的操作也不被汇编成机器代码，只能为汇编程序所识别并指导汇编如何进行。 伪指令语句是相对指令语句而言的。

指令语句
每一条指令语句在源程序汇编时都要产生可供 MCU 执行的指令代码（即目标代码），所以这种语句又叫可执行语句。每一条指令语句表示 MCU 具有的一个基本能力，如数据传送，两数相加或相减，移位等，而这种能力是在目标程序（指令代码的有序集合）运行时完成的，是依赖于片内的中央处理器（CPU）、存储器、I / O 接口等硬件设备来实现的。

伪指令语句
伪指令语句是用于指示汇编程序如何汇编源程序，所以这种语句又叫命令语句。例如源程序中的伪指令语句告诉汇编程序：该源程序如何分段，有哪些逻辑段在程序段中哪些是当前段，它们分别由哪个段寄存器指向；定义了哪些数据，存储单元是如何分配的等等。伪指令语句除定义的具体数据要生成目标代码外，其它均没有对应的目标代码。伪指令语句的这些命令功能是由汇编程序在汇编源程序时，通过执行一段程序来完成的，而不是在运行目标程序时实现的。
伪指令表可参考附录III。

## 3.3 包含文档

### INCLUDE:

语法： INCLUDE FILE

说明：

该命令后面需要跟一个程序文件，如果该程序文件不在当前正在编译的程序目录下，用户就需要为其指定路径。该程序文件的语法和一般的文件无异。通常，扩展名是.H 的文件一般包括常数和宏的定义，扩展名是.asm 则包含了指令。被包含的档内还可以使用 INCLUDE 命令，嵌套最多 255 层。

例如：

```
INCLUDE    SN88X.H              // 包含自定义的变量名
INCLUDE    C: \PROJECT.H        // 包含自定义的宏
INCLUDE    sub \ Filename.ASM
INCLUDE    ..\Parent\File2.ASM
```

### INCLUDESTD:

语法：INCLUDESTD FILE

说明：

该命令后面需要一个程序文件，路径已经被固定为SN8ASM.EXE 所在路径，其余参考INCLUDE 命令。此命令可以使用户方便取用系统提供的宏文件。

例：

```
INCLUDESTD    MACRO1.H   // 包含系统常用宏
INCLUDESTD    MACRO2.H   // 包含系统常用宏文件
```

### INCLUDEBIN:

语法：INCLUDEBIN FILE

说明：

该命令后面需要一个BIN 文件，如果该文件不在当前正在编译的程序目录下，用户就需要为其指定路径名。该文件的数据是以WORD 为单位。不足一个WORD 的，按一个WORD 计。

例如：

```
INCLUDE    SPEECH.BIN
INCLUDE    C: \SOUND.SND
```

# 3.4 宏

SONiX 支持下列默认的宏，利用宏可以更容易的编写程序，下面的宏都在编译器目录下的MACRO1.H，MACRO2.H 和MACRO3.H 三个档中有定义。

| 分类 | 汇编助记码 | 扩展格式 | 展开后指令条数 | 功能 | 标志 | | | 周期 |
|---|---|---|---|---|---|---|---|---|
| | | | | | CF | DC | ZF | |
| COMMAND | CLC | B0BCLR FC | 1 | 清C 标志 | 0 | – | – | 1 |
| | STC | B0BSET FC | 1 | 置C 标志 | 1 | – | – | 1 |
| | RSTWDT | B0BSET FWDRST | 1 | 复位看门狗计数器 | – | – | – | 1 |
| | EINT | B0BSET fgie | 1 | 全局中断位使能 | – | – | – | 1 |
| | DINT | B0BCLR fgie | 1 | 全局中断位禁止 | – | – | – | 1 |
| | NOT A | XOR A, #0FFh | 1 | A ← ~ A | – | – | – | 1 |
| | NEG A | XOR A, #0FFh<br>ADD A, #1 | 2 | A ← – A | – | – | – | 2 |
| ROTATE/SHIFT | SHL memory | B0BCLR FC<br>RLCM memory | 2 | memory ←memory * 2 | – | – | – | 2 |
| | SHR memory | B0BCLR FC<br>RRCM memory | 2 | memory ←memory / 2 | – | – | – | 2 |
| | B2B bit1, bit2 | B0BCLR bit2<br>B0BTS0 bit1<br>B0BSET bit2 | 3 | 交换bit2和bit1的状态 | – | – | – | 3 |
| | ROL mem | RLCM mem<br>B2B FC, mem.0 | 4 | FC ← mem.7<br>mem←mem * 2 + FC | – | – | – | 4 |
| | ROR mem | RRCM mem<br>B2B FC, mem.7 | 4 | FC ← mem.0<br>Mem←mem / 2 +FC * 80h | – | – | – | 4 |
| | RCR mem | RRCM mem | 1 | 另一种写法 | – | – | – | 1 |
| | RCL mem | RLCM mem | 1 | 另一种写法 | – | – | – | 1 |
| BRANCH | JZ address | B0BTS0 FZ<br>JMP address | 2 | 如果ZF == 1，就跳到address | – | – | – | 2 |
| | JNZ address | B0BTS1 FZ<br>JMP address | 2 | 如果ZF == 0，就跳到address | – | – | – | 2 |
| | JC address | B0BTS0 FC<br>JMP address | 2 | 如果CF == 1，就跳到address | – | – | – | 2 |
| | JNC address | B0BTS1 FC<br>JMP address | 2 | 如果CF == 0，就跳到address | – | – | – | 2 |
| | JDC address | B0BTS0 FDC<br>JMP address | 2 | 如果DCF == 1，就跳到address | – | – | – | 2 |
| | JNDC address | B0BTS1 FDC<br>JMP address | 2 | 如果DCF == 0，就跳到address | – | – | – | 2 |
| BRANCH | JB1 bit, addr | BTS0 bit<br>JMP addr | 2 | 如果bit == 1，就跳到addr | – | – | – | 2 |
| | JB0 bit, addr | BTS1 bit<br>JMP addr | 2 | 如果bit == 0，就跳到addr | – | – | – | 2 |
| | DJNZ mem, adr | DECMS mem<br>JMP adr | 2 | mem ← mem –1 如果 mem != 0，就跳到adr | – | – | – | 2-3 |
| | IJNZ mem, adr | INCMS mem<br>JMP adr | 2 | mem ← mem +1 如果 mem != 0，就跳到adr | – | – | – | 2-3 |
| | CJNE A, m, adr | CMPRS A, m<br>JMP adr | 2 | 如果ACC != m，就跳到adr | – | – | – | 2-3 |
| | CJE A, m, adr | CMPRS A, m<br>JMP $+2<br>JMP adr | 3 | 如果ACC == m，就跳到adr | – | – | – | 3-4 |
| | CJAE A, m, adr | CMPRS A, m<br>B0BTS0 FC<br>JMP adr | 3 | 如果ACC >= m (Above Equal)，就跳到adr | – | – | – | 3-4 |
| | CJAE m, A, adr | CMPRS A, m<br>B0BTS1 FC<br>JMP adr | 3 | 如果m >= ACC (Above Equal)，就跳到adr | – | – | – | 3-4 |
| | CJBE A, m, adr | CJAE m, A, adr | 3 | 如果ACC <= m (Below Equal)，就跳到adr | – | – | – | 3-4 |
| | CJBE m, A, adr | CJAE A, m, adr | 3 | 如果m <= ACC (Below Equal)，就跳到adr | – | – | – | 3-4 |
| | CJA A,m, adr | CMPRS A, m<br>B0BTS1 FC<br>JMP $+2<br>JMP adr | 4 | 如果ACC > m (Above)，就跳到adr | – | – | – | 4-5 |
| | CJA m,A adr | CMPRS A, m<br>B0BTS0 FC | 4 | 如果m > ACC (Above)，就跳到adr | – | – | – | 4-5 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | JMP $+2<br>JMP adr | | | | | | |
| CJB A,m, adr | CJA m, A, adr | 4 | 如果ACC < m (Below)，就跳到adr | – | – | – | 4-5 | |
| CJB m,A adr | CJA A, m, adr | 4 | 如果m < ACC (Below)，就跳到adr | – | – | – | 4-5 | |

MACRO:

语法：

```
NAME    MACRO    [PARA1[，PARE2，…] ]
…
ENDM
```

说明：

使用宏可以简化程序的撰写，使用子程序可以缩短程序代码的长度。如果善用宏和子程序，可以使程序达到最佳状态。宏的参数不能超过255 个，长度不限，可以由A~Z，a~z，0～9 和@#_.$ 等组成，如果必须使用特殊字符，则要用符号< >将特殊字符分开。此外，宏中还可以嵌套其它宏，且层数不限。

例：

```
MOV_    MACRO  ADDRESS,  VALUE
MOV     A, VALUE                       // 宏内容
MOV     ADDRESS, A
ENDM

MOV_    1, #1 //  使用宏
MOV_    2, <#6*8+1>
```

EXPAND:

语法： 同上说明：在编译出现错误或仿真时，使用MACRO 将使光标停在调用宏的位置，使用EXPAND 会使光标停在宏指令内部，这有利于错误检查。

REPEAT:

语法：

```
REPEAT    COUNT
…         ；REPEATED    COMMAND
ENDM
```

说明：

用REPEAT 重复执行一段命令。重复的次数由COUNT 值而定。用ENDM 表示此命令段结束，用REPEAT 表示命令段的开始。

例如：

```
RRCM_1  MACRO          MEMORY,  VALUE
REPEAT  VALUE
RLCM    MEMORY
ENDM
ENDM
…
RRCM_     0, 4                 ；RAM[0] 的内容被右移4次
```

FOR:

语法：FOR PARAMETER， < PARAMETER 1, PARAMETER 2 …>

```
…
ENDM
```

说明：

用FOR重复执行一段命令。重复的次数视尖括号内的参数而定，依照逗号分开的参数顺序，一次以一个参数取代PARAMETER的值。

例如：

```
EMP  ＝ 0
FOR       I, <M0, M1, M2, M3, M4, M5>
I EQU     TEMP
EMP  ＝   TEMP+1
ENDM
                              ；M0/1/2/3/4/5 EQU      0/1/2/3/4/5
```

FORC:

语法：　FORC　PARAMETER, <WORD SERIAL>

```
        …           ；要被重复的命令
        ENDM
```

说明：

用FORC 重复执行一段命令。重复的次数视尖括号内的字符串的长度而定，依照字符串的字符顺序，一次以一个字符取代PARAMETER 的值。

例如：

```
FORC    I, <012345>
M&I     EQU         I
ENDM
```

EXITM:

语法：　EXITM

说明：

用EXITM 命令可以使宏展开提前结束，常在宏指令出错时使用。

例如：

```
MEM         MACRO       value
.IF         value>=10
            ERROR       value must < 10
            EXITM
.ENDIF
            M&value     EQU   value
            END
```

## 宏指令内的运算符

&

语法：…&PARAM

说明：在宏指令内，使对应的字符串转换成参数。

例如：

```
FORC    I, <012345>
M&I       EQU     I                 ; M0/1/2/… EQU   0/1/2/…
ENDM
```

%

语法：%PARAM 说明：在宏指令内，使对应的字符串转换成数值。例如：

```
ERROR_    MACRO         E1   E2   E3
ERROE   E1 E2   E3
ENDM
TEMP  ＝     10
ERROR_ TEMP IS %TEMP
```

则编译后，会显示TEMP IS 0XA 的错误信息。

!

语法：! 字符

说明：在宏指令内部，使下一个字符不含任何特殊意义。

例如：

```
ENUM      MACRO      CH
FORC     I, <012345>
CH!&I      EQU      I               ; CH&0/1/… EQU        0/1/…
ENDM
ENDM
```

```
ENUM    M                    ; M0/1/2/… EQU 0/1/2/…
```

上面的例子使用了两层宏指令，'ENUM M '被展开成：

```
FORC    I, <012345>
M&I     EQU   I              ; M0/1/2…    EQU    0/1/2/…
ENDM
```

如果不使用！，则 & 将在第一次展开时被使用。

```
FORC    I, <012345>
M&I     EQU   I              ; M0/1/2…    EQU    0/1/2/…
ENDM
```

而第二次展开时，已经是变量MI 了。

## LOCAL

宏指令内部支持局部标号。例如：

```
        XXX      MACRO
        LOCAL    XX1, XX2
        JMP      XX1
XX1:
        JMP      XX2
XX2:
        NOP
        ENDM
```

XXX 可以多次调用，但XX1, XX2 不可以重复定义。

## 缺省和变量参数

看下面的宏，当您缺少第一个参数时，并不会产生任何错误。

```
ADD2    MACRO    A, B
        DW       A+B
        ENDM
ADD2    4                    ; DW+4
```

当您希望第一个参数不要省略时，则可改写为如下：

```
ADD2    MACRO    A: REQ, B
        DW       A+B
        ENDM
ADD     2 , 4                ; 错误产生
```

又如果您希望第二个参数可以省略，而用默认值时，可改写为：

```
ADD2    MACRO    A: REQ, B: =0;  或  B: =< 表达式>
        DW   A+B
        ENDM
ADD2    4        ;DW    4+0
```

最后，若您希望是一个变量，则可改写为如下：

```
ADDN    MACRO    params: VARARG
        TEMP = 0
        FOR      I, <params>
        TEMP = TEMP+I
        ENDM
        DW   TEMP
        ENDM
ADDN    1, 2, 3, 4   ; DW    10
```

## 3.5 条件编译控制

语法：

```
IF            expression         ; 不为0时为真
IFE           expression         ; 为0时为真
IFB           <argue>            ; 为空白时为真
IFNB          <argue>            ; 不为空白时为真
IFDEF         symbol             ; symbol被定义时为真
IFNDEF        symbol             ; symbol不被定义时为真
IFIDN         <str1>, <str2>     ; str1 == str2 时为真
IFDIF         <str1>, <str2>     ; str1 <> str2 为真
IFIDNI        <str1>, <str2>     ; str1 == str2 ( 不分大小写)时为真
IFDIFI        <str1>, <str2>     ; str1 <> str2 ( 不分大小写)时为真
ELSEIF        expression
ELSEIFE       expression
ELSEIFB       <argue>
ELSEIFNB      <argue>
ELSEIFDEF     symbol
ELSEIFNDEF    symbol
ELSEIFIDN     <str1>, <str2>
ELSEIFIDNI    <str1>, <str2>
ELSEIFDIF     <str1>, <str2>
ELSEIFDIFI    <str1>, <str2>
ELSE          ENDIF
```

说明：

用IF 命令检查具体条件及控制程序的编译，并用ENDIF 命令结束，还可加入选择性的ELSEIF/ELSE 命令，并支持嵌套。

例如：

为避免参数空白，可使用如下语法：

```
 TEMP = 0
  FOR       I <ZERO ONE THREE>
  IFNB      <I>
  I         EQU     TEMP
  ENDIF
  TEMP =    TEMP+1
  ENDM
```

为避免.H 文件被重复加载，应使用如下语法：

```
IFNDEF          —TESTFILE—          ; 加在程序的开始
—TESTFILE—      EQU      1          ; 程序内容
…
ENDIF                               ; 程序结束
```

还可以使用嵌套：

```
IF   VAR== 1
…
    ELSEIF   VAR     <= 10
…
IF   VAR >5
…
    ELSE
…
    ENDIF
…
    ELSEIF VAR    <= 100
…
    ELSE
…
    ENDIF
```

## 条件编译下的错误显示

语法：

```
.ERR                                          ；强制产生错误
.ERRNZ        expression                      ；不为0 时产生错误
.ERRE         expression                      ；为0 时产生错误
.ERRB         <argue>                         ；为空白时产生错误
.ERRNB        <argue>                         ；不为空白时产生错误
.ERRDEF       symbol                          ；symbol 被定义时产生错误
.ERRNDEF      symbol                          ；symbol 不被定义时产生错误
.ERRIDN       <str1>, <str2>                  ；tr1==str2 时产生错误
.ERRDIF       <str1>, <str2>                  ；str1< >str2 时产生错误
.ERRIDNI      <str1>, <str2>                  ；str1==str2 ( 不分大小写)时产生错误
.ERRDIFI      <str1>, <str2>                  ；str1< >str2 ( 不分大小写)时产生错误
.ERROR        <string>                        ；强制产生错误，并以string 为错误信息
.ECHO         <string>                        ；产生string 为错误信息
```

说明：使用这些条件编译命令可以检测一些特定的条件，并且可以输出错误信息以提醒用户。

例：

```
ECHO          Here be compiled                // 当编译到此行时会显示信息
```

对下面宏定义我们可以改变一下它的表达方式。

```
ADD2          MACRO                           A: REQ,     B: = <0>
              DW                              A+B
              ENDM
```

新的表达方式：

```
ADD2          MACRO                           A, B
              .ERRB                           <A>
              IFB                             <B>
              DW                              A
              ELSE
              DW                              A+B
              ENDIF
              ENDM
```

显然这并不是一个很好的方式，只是个示范。

## .LIST:

语法： **.LIST**

说明：

利用.LIST 可以把LIST 命令以下的源程序包含在列表文件中，这是默认设定值。

例如：

```
.LIST
…                                             ；将出现在列表文件中
.NOLIST
```

## .NOLIST:

语法：**.NOLIST**

说明：

用 .NOLIST 可以关闭列表功能，直到另一个.LIST 将它打开。

例如：

```
.NOLIST
…                                             ；将不会出现在列表档中
.LIST
```

## .LISTIF:

语法：**.LISTIF**

说明：

用 .LISTIF 列表条件编译命令，即使命令没有被编译过，在列表档中也会列出这些命令。

例：

```
.LISTIF IF 0
 …                                          ；将出现在列表档中
ENDIF
```

## .NOLISTIF:

语法：**.NOLISTIF**

说明：

用.NOLISTIF 来隐藏没有被编译过的指令，这是默认设定值。

例：

```
.NOLISTIF IF 0
 …                                          ；将不会出现在列表文件中
ENDIF
```

## .LISTMACRO:

语法：**.LISTMACRO**

说明：

用.LISTMACRO 列出宏内部能产生程序代码或数据的命令，这是默认设定值。

例如：

```
.LIST      MACRO     TEMP =0
           REPEAT 8
           DW    TEMP                  ；将出现在列表档中
           DEMP = TEMP+1               ；不会出现在列表档中
           ENDM
```

## .NOLISTMACRO:

语法：**.NOLISTMACRO**

说明：

用.NOLISTMACRO 将使宏指令内部的代码不被扩展。

例如：

```
.NOLIST    MACRO    TEMP = 0
           REPEAT   8
```

## .LISTMACROALL:

语法：**.LISTMACROALL**

说明：

用.LISTMACROALL 列出宏指令中的所有指令。

例如：

```
.LIST    MACROALL    TEMP = 0
         REPEAT 8
         DW    TEMP                  ；将出现在列表档中
         DENP = TEMP+1               ；将出现在列表档中
         ENDM
```

# 附录

## 附录 I 编译器错误信息说明

附表 I -1 错误信息一览表

| 编号 | 名称 | 提示 | 代表的含义 |
|---|---|---|---|
| 1 | Syntax_Error | Syntax error happen !!! | 语法错误 |
| 2 | Detrop_SEnd | Not Need ) but find !!! | 多出右括号 |
| 3 | Miss_SStart | Need ( but not find !!! | 缺少左括号 |
| 4 | Miss_SEnd | Need ) but not find !!! | 缺少右括号 |
| 5 | Detrop_LEnd | Not Need } but find !!! | 多出右大括号 |
| 6 | Miss_LEnd | Need } but not find !!! | 缺少右大括号 |
| 7 | Detrop_MarkEnd | Not Need */ but find !!! | 多出 '*/' 符号 |
| 8 | Miss_MarkEnd | Need */ but not find !!! | 缺少 '*/' 符号 |
| 9 | Need_Space | Need space to next symbol !!! | 符号之间需要空隔 |
| 10 | Detrop_Space | Not need space here !!! | 此不需要间隔 |
| 11 | Detrop_Char | Not need char here !!! | 此不需要字元 |
| 12 | Need_Value | Here need a value !!! | 缺少运算元 |
| 13 | Bad_Value | Invalide value !!! | 无效值 |
| 14 | Bad_Char | Need format 'X' here !!! | 此需要字元格式 |
| 15 | No_Bit | Invalid bit operation !!! | 无效的位元运算 |
| 16 | Need_Bit | Here need format : XX.bit !!! | 运算元需是 'XX.bit' 型态 |
| 17 | Need_Imm | Here need format : #xx !!! | 运算元需是 '#XX' 型态 |
| 18 | No_Operator | Here +-*/!~() HIGH/LOW/MID can't access !!! | 运算子 +-*/!~() 对 HIGH/MID/LOW 不能存取 |
| 19 | Bad_HMLJ | only Label$H/M/L/J can access !!! | 只有 Label$H/M/L/J 可以存取 |
| 20 | Bad_HML | only Label$H/M/L can access !!! | 只有 Label$H/M/L 可以存取 |
| 21 | Attrib_Order_Error | HIGH/MID/LOW must before -,!,~ !!! | 运算子 -,!,~ 必须置于 HIGH/MID/LOW 之前 |
| 22 | Attrib_Repeat_Error | HIGH/MID/LOW repeat too much !!! | HIGH/MID/LOW 重复太多 |
| 23 | Bad_String | Expect \"string..\" here !!! | 此缺少字串 |
| 24 | UnAccess_Symbol | Symbol not access here !!! | 符号无法对此存取 |
| 25 | Undef_Symbol | Undefined Symbol !!! | 没有定义的符号 |
| 26 | Unknow_Command | Unknown Command !!! | 未知的指令 |
| 27 | Over_0x100 | The value must limit 0 - 0xFF !!! | 此值必须限制在 0x00 - 0xFF 范围 |
| 28 | Over_0x10000 | The value must limit 0 - 0xFFFF !!! | 此值必须限制在 0x0000 - 0xFFFF 范围 |
| 29 | Over_Range | The value over range !!! | 值超过范围 |
| 30 | Over_Imm_Range | The #nn value over range !!! | #nn(立即值)超过范围 |
| 31 | Over_Bit1_Range | The value (MM.b), MM value over range !!! | RAM 的值超过范围 |
| 32 | Over_Bit2_Range | The value (mm.B), B value over range !!! | RAM 的 Bit 值超过范围 |
| 33 | Code_OverWrite | The Code has overwrite from here, please check it !!! | 程式码会从此覆写,请检查 |
| 34 | Data_OverWrite | The Data has overwrite from here, please check it !!! | 资料会从此覆写,请检查 |
| 35 | Over_Code | Over Program-Code Range !!! | 超过 User ROM Size 大小 |
| 36 | Over_Voice | Over ROM Code Range !!! | 超过 ROM Size 大小 |
| 37 | Over_RAM | Over RAM Range !!! | 超过 RAM Size 大小 |
| 38 | Jmp_Over | JMP, CALL, ... command over range !!! | JMP, CALL, ... 指令超过范围 |
| 39 | Over_OJump | Over the JUMP range !!! | 超过 JUMP 指令范围 |
| 40 | Play_Over | PLAY command over range !!! | PLAY 指令超过范围 |
| 41 | Over_List | The List buffer has over size, force disable list !!! | List buffer 已超过范围 ,强制禁能 list |
| 42 | Over_Stack | Over Stack, Please check your program for loop-include !!! | 堆迭溢位,请检查程式中 |
| 43 | Over_TextEqu | Over TextEqu, Please check your program for loop-textequ !!! | TextEqu Over,请检查程式中 textequ 定义 |
| 44 | Over_Buf2 | Over Stack, Please check your program for loop-macro !!! | 堆迭溢位,请检查程式中巨集回圈 |
| 45 | Buffer_Over | Buffer over range to record data !!! | Buffer 超过纪录 data 的范围 |

| | | | |
|---|---|---|---|
| 46 | Inter_Error | Happen internal unknow error !!! | 发生内部未知的错误 |
| 47 | INI_Error | .INI happen error, stop read !!! | .INI 档发生错误,停止读取 |
| 48 | Source_Error | Source error, stop compile !!! | Source 错误,停止编译 |
| 49 | File_Null | File not exist, stop compile !!! | 档案不存在,停止编译 |
| 50 | No_Chip | missing CHIP SNxxx at head of program !!! | 缺少 chip name 宣告 |
| 51 | Chip_Fail | Wrong device selection !!! | chip name 宣告错误 |
| 52 | Chip_Redefined | Chip select fail, can't redefined chip !!! | Chip 选择失败,无法重新定义 chip |
| 53 | Not_In_System | The command only be supported at system !!! | 此指令只支援此系统 |
| 54 | Not_The_uC | The command not be supported at the serial uC !!! | 此指令不支援此系列 MCU |
| 55 | ICE_Break_Format | Here need format : .Command reg1, reg2 ... !!! | 此需要.Command reg1, reg2 …格式 |
| 56 | Align_Format | Only access .ALIGN number, number = 2, 4, 8,.., 65536 !!! | 只存取区段对齐(.ALIGN )值,对齐值等于 2,4,8..,65536 |
| 57 | Text_Format | Must use the format .TEXT { .. } !! | 必须使用 .TEXT { .. }格式 |
| 58 | Instr_Format | At INSTR & SUBSTR, the start pos must >= 1 !!! | INSTR & SUBSTR 起始位置必须大于或等于 1 |
| 59 | Miss_Colon | Need : but not found !!! | 缺少 : 符号 |
| 60 | Miss_Dot | Need , but not found !!! | 缺少 , 符号 |
| 61 | Miss_SDot | Need . but not found !!! | 缺少 . 符号 |
| 62 | Miss_equ | Need = but not found !!! | 缺少 = 符号 |
| 63 | Miss_dequ | Need := but not found !!! | 缺少 := 符号 |
| 64 | Miss_ALU | Need A but not found !!! | 缺少 ACC Register |
| 65 | Bad_AA | Only @@: be accessed !!! | 只有 @@: 可存取 |
| 66 | Loss_AA | Need @@: but lost !!! | 缺少 @@: 符号 |
| 67 | Label_Redef | The Label has exist !!! | 标号已经存在 (标号重复) |
| 68 | Symbol_Redef | The Symbol has been defined !!! | 符号已经定义 (符号重复) |
| 69 | Symbol_BadType | The Symbol has other type !!! | 此符号有其他型态 |
| 70 | Symbol_Label | The Symbol must be label or set by EQU !!! | 此符号必须是标号或由 EQU 设定 |
| 71 | Resource_Redef | The Resource has exist !!! | 此资源档已存在 |
| 72 | Resource_NoUse | The Resource file not be used !!! | 此资源档不能使用 |
| 73 | Resource_Error | Loss the Resource or translate fail !!! | 遗失资源档或转移错误 |
| 74 | Bad_Resource | Unexpect filename, please define before PROGRAM !!! | 非预期档名,请在 PROGRAM 之前定义 |
| 75 | Resource_NoFind | Resource file not be find !!! | 找不到资源档 |
| 76 | Resource_NoTran | Resource file translate fail !!! | 资源档转移错误 |
| 77 | Macro_Error | Macro error, stop compile !!! | 巨集错误,停止编译 |
| 78 | MacroName_Error | Macro_Name error, stop compile !!! | 巨集名称错误,停止编译 |
| 79 | Macro_Redef | MACRO redefined !!! | 巨集重复定义 |
| 80 | Loss_Param | Macro request parament loss !!! | 遗失巨集需要的参数 |
| 81 | Tran_Param_Val | Check fail for translate parament to value !!! | 检查参数传值失败 |
| 82 | EXITM_Fail | EXITM must be placed in MACRO/REPEAT/FOR/FORC .. ENDM !!! | EXITM 必须置于 MACRO/REPEAT/FOR/FORC .. ENDM 之中 |
| 83 | TextEqu_Error | Define TEXT must be used as <...> !!! | 必须使用 <...>定义 TEXT |
| 84 | TextEqu_BeUsed | TEXTEQU be used before defined !!! | TEXTEQU 是使用在定义之前 |
| 85 | TextEqu_Assign | TEXT must be assigned by TEXTEQU, CATSTR, ... !!! | TEXT 必须由 TEXTEQU, CATSTR, ... 指定 |
| 86 | Need_TextStr | Need TEXT or <..> after TEXTEQU, CATSTR, ... !!! | TEXTEQU, CATSTR, …之后须是 TEXT 或 <..> |
| 87 | Condiction_Error | Condiction error, stop compile !!! | 条件错误停止编译 |
| 88 | Detrop_ELSE | Not need ELSE here !!! | 多出 "ELSE" 指令 |
| 89 | Detrop_ELSEIF | Not need ELSEIF here !!! | 多出 "ELSEIF" 指令 |
| 90 | Detrop_ENDIF | Not need ENDIF here !!! | 多出 "ENDIF" 指令 |
| 91 | Loss_ENDIF | Need ENDIF but not found !!! | 缺少 "ENDIF" 指令 |
| 92 | ERR_Force | Forced error | 强制错误 |
| 93 | ERR_True | Forced error - expression false(not 0) | 强制错误 - 运算式为假(不等于零) |
| 94 | ERR_False | Forced error - expression true(0) | 强制错误 - 运算式为真(等于零) |
| 95 | ERR_Blank | Forced error - string blank | 强制错误 - 字串空白 |

| | | | |
|---|---|---|---|
| 96 | ERR_NBlank | Forced error - string not blank | 强制错误 - 字串不是空白 |
| 97 | ERR_Define | Forced error - symbol defined | 强制错误 - 符号定义 |
| 98 | ERR_NDefine | Forced error - symbol not defined | 强制错误 - 符号未定义 |
| 99 | ERR_Dif | Forced error - string different | 强制错误 - 字串不同 |
| 100 | ERR_Idn | Forced error - string identical | 强制错误 - 字串一致 |
| 101 | LOG_ANDOR | Only && \|\| can be use for logic operator | 逻辑运算只能使用 &&及 \|\| |
| 102 | LOG_L00 | missing '(' before ')' | 右括号之前缺少左括号 |
| 103 | LOG_L01 | missing ')' before EOL | EOL 之前缺少右括号 |
| 104 | LOG_L02 | missing ')' before THEN/GOTO | THEN/GOTO 之前缺少右括号 |
| 105 | LOG_AELSE | missing .IF before .ELSExx / .ENDIF | .ELSExx / .ENDIF 之前缺少 IF |
| 106 | LOG_BENDIF | missing .ENDIF before EOF | EOF 之前缺少.ENDIF |
| 107 | LOG_AENDW | missing .WHILE before .ENDW | .ENDW 之前缺少.WHILE |
| 108 | LOG_AUNTIL | missing .REPEAT before .UNTIL / .UNTILDZ / .UNTILIZ | .UNTIL / .UNTILDZ / .UNTILIZ 之前缺少.REPEAT |
| 109 | LOG_ASWITCH | missing .SWITCH before .CASE .DEFAULT .ENDS | .CASE .DEFAULT .ENDS 之前缺少 .SWITCH |
| 110 | LOG_ABREAK | missing .WHILE .REPEAT .SWITCH before .BREAK | .BREAK 之前缺少.WHILE .REPEAT .SWITCH |
| 111 | LOG_ACONTINUE | missing .WHILE .REPEAT before .CONTINUE | .CONTINUE 之前缺少.WHILE .REPEAT |
| 112 | LOG_GTLT | Use 'mem/acc >,>=,<,<=,==,!= mem/imm' for logic operator | 需使用 '>,>=,<,<=,==,!= '逻辑运算子 |
| 113 | LOG_PATTERN | Syntax Error, Ex : .CMD 'bit && mem > imm \|\| !bit' | 语法错误,例如 .CMD 'bit && mem > imm \|\| !bit' |
| 114 | LOG_NOTBIT | only '! bit' can be access for logic operator | 逻辑运算子只有 '! bit'可以存取 |
| 115 | LOG_IFGO | only '.IF log THEN/GOTO xxx' can be access for logic operator | 逻辑运算子只有 '.IF log THEN/GOTO xxx' 可以存取 |
| 116 | LOG_IFTHEN | for logic '.IF log THEN cmd', the 'cmd' must 1 word | 于逻辑判断 '.IF log THEN cmd'式,其中'cmd'必须是 1 个 word |
| 117 | LOG_GT255 | Use 'mem/acc > #255' or 'mem/acc <= #255' exist logic question | 使用'mem/acc > #255' 或 'mem/acc <= #255'运算式有逻辑问题 |
| 118 | LOG_LT0 | Use 'mem/acc < #0 ' or 'mem/acc >= #0' exist logic question | 使用 'mem/acc < #0 ' 或 'mem/acc >= #0'运算式有逻辑问题 |
| 119 | OP_NEED_INT | the expression can't access label or reg for operator | 运算式中的运算子是不能对标号或暂存器做运算 |
| 120 | Public_Syntax | Need format : Public name1 [,name2 ..] | 需要 Public name1 [,name2 ..]格式 |
| 121 | Public_Keyword | Can not use keyword for public name !!! | 公用名称不能使用关键字 |
| 122 | Public_Redefined | The public name has been declared !!! | 公用名称已宣告 |
| 123 | Extern_Syntax | Need format : Extern name1 [,name2 ..] | 需要 Extern name1 [,name2 ..]格式 |
| 124 | Extern_Redef | Extern name has other type !!! | Extern 名称有另一种型态 |
| 125 | Bug_B0MOV_86 | The command B0MOV M, #I, but #I can't be #0E6, #0E7h | B0MOV M, #I 指令,#I 不能是#0E6h,#0E7h |
| 126 | Bug_S8387P | SN8P1602, SN8P1603 and SN8P1604 has following DW limitation: | SN8P1602, SN8P1603 及 SN8P1604 有下面 DW(Define Word)限制 |
| | | If high byte = 0 then the bit 0 and bit 1 of low byte must be zero | 如果高位元组等于零,则低位元组的 bit0 及 bit1 必须为零 |
| 127 | Bug_ICE_AF | Instant addressing mode can not use #0AFh, please call SONIX FAE | 立即定址模式不能使用#0AFh,请联络 SONIX FAE |
| 128 | Need_AluImm | Here need parament : ALU or #xx !!! | 运算元需要 ALU or #xx 型态的参数 |
| 129 | Msg_AsmRETI | Before RETI, please use <POP> <B0XCH A, m> to protect FZ | 使用 RETI 指令之前,请使用<POP> <B0XCH A, m>指令来保护 FZ |
| 130 | Msg_AddPCL | The JMP command over 256 boundary | JMP 指令超过 256 范围 |
| 131 | Chk_Code_0_3 | At 0.. 3, no find JMP (>=8) command !!! | 在 0..3 位址没有找到 JMP(>=8)指令 |
| 132 | Chk_Code_4_7 | The Code Area 4..7 be reserved for Test !!! | 程式码区 4..7 位址留作测试专用 |
| 133 | Chk_Option_Loss | The code_option loss following items !!! | Code Option 遗失下面项目 |
| 134 | Chk_Option_Pos1 | .Code_Option must be declared after CHIP defined !! | .Code_Option 必须宣告在 CHIP 定义之后 |
| 135 | Chk_Option_Pos2 | The code_option can't set at here !! | code option 不能在此设定 |
| 136 | Chk_Option_Redef | The code_option has redefined !! | code option 重复定义 |
| 137 | Chk_Option_NoDef1 | The code_option not be defined !! | code option 没有定义 |

| | | | |
|---|---|---|---|
| | | Please use the following code to defined .Code_Option | 请使用下面的程式码定义 code option |
| 138 | Chk_Option_NoDef2 | The code_option not be defined !! | code option 没有定义 |
| 139 | Opt_HighClk_2 | At RC mode, HighClk div 2 must Enable !! | 在 RC 模式下,必须致能 HighClk div 2 |
| 140 | Opt_OSG_Enable | At slow crystal, OSG must Enable !! | 在低频的模式下,必须致能 OSG |
| 141 | Over_IP_Bound | The code may be over 16K boundary, please check it !!! | 请检查程式码是否超过 16K 范围 |
| 142 | Over_0x8087 | The value must limit 0x80 - 0x87 !!! | 此值必须限制在 0x80 - 0x87 范围 |
| 143 | Over_0x808F | The value must limit 0x80 - 0x8F !!! | 此值必须限制在 0x80 - 0x8F 范围 |
| 144 | RAM_ReadOnly | This is read-only RAM .. | 此 RAM 只能 read |
| 145 | RAM_WriteOnly | This is write-only RAM .. | 此 RAM 只能 write |
| 146 | RAM_NoUse | The RAM can't use .. | 此 RAM 不开放使用 |
| 147 | BIT_ReadOnly | This is read-only BIT .. | 此 BIT 只能 read |
| 148 | BIT_WriteOnly | This is write-only BIT .. | 此 BIT 只能 write |
| 149 | BIT_NoUse | The BIT can't use .. | 此 BIT 不开放使用 |
| 150 | BIT_MoveOnly | Only can use MOV command to change the bit .. | 只能使用 MOV 指令改变 bit 状态 |
| 151 | Roll_Pos | .Rolling_Code command only be placed at .CODE segment !!! | Rolling Code command 只能在 .code 程式区宣告 |
| 152 | Roll_Cnt | The maximum size of Rolling Code is 4 words !!! | Rolling Code 最大支援 4 words |
| 153 | Chip_Format | only <CHIP xx, PIC> can be access | 只有<CHIP xx, PIC>可以存取 |
| 154 | AsmOpt_Format | .Assembly Option num title, str0, str1 !! | inc file 中 code option 格式有错误 |
| 155 | PicAsm1_Format | Only [ Command Mem, W/F ] can access !! | 只有[ Command Mem, W/F ]可以存取 |
| 156 | PicAsm2_Format | Only [ Command Mem, Bit ( < 8) ] can access !! | 只有[ Command Mem, Bit ( < 8) ]可以存取 |
| 157 | PicAsm3_Format | Only 0xD1 .. 0xD7 can access !! | 只有 0xD1..0xD7 可以存取 |

# 附录 II 菜单命令，工具和快捷方式一览表

下面列出了 M2IDE 集成开发环境下的菜单中各项命令、工具栏图标、默认快捷键以及它们的说明。

文件菜单和文件命令

附表 II-1 文件菜单和文件命名 File

| File 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| New | | Ctrl+N | 创建一个新的源文件或文本文件 |
| Open | | Ctrl+O | 打开已有文件 |
| Close | | | 关闭当前的文件 |
| Save- | | Ctrl+S | 保存当前的文件 |
| Save As… | | | 保存并重新命名当前文件 |
| New Project | | | 新建一个工程 |
| Open Project | | | 打开一个已有的工程 |
| Close Project | | | 关闭当前的工程 |
| Recent Project | | | 指明当前工程 |
| 1~9 | | | 打开最近使用的源文件或文本文件 |
| Exit | | | 退出并提示保存文件 |

编辑菜单和编辑命令

附表 II-2 编辑菜单与编辑器命令

| Edit 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| Undo | | Ctrl+Z | 撤销上一次操作 |
| Redo | | Ctrl+Y | 重做上一次撤销的命令 |
| Cut | | Ctrl+X | 将选中的文字剪切到剪贴板 |
| Copy | | Ctrl+C | 将选中的文字复制到剪贴板 |
| Paste | | Ctrl+V | 粘贴剪贴板的文字 |
| Select All | | Ctrl+A | 全选 |
| Find | | Ctrl+F | 在当前文件中查找文字 |
| Find in Files | | | 在几个文件中查找文字 |
| Replace | | Ctrl+H | 替换特定文字 |
| Prev BookMark | | F2 | 将光标移到上一个书签 |
| Next BookMark | | Shift+F2 | 将光标移到下一个书签 |
| Toggle BookMark | | Ctrl+F2 | 在当前行放置书签 |
| Clear BookMarks | | Ctrl+Shift+F2 | 清除当前文件中的所有书签 |

视图菜单

附表Ⅱ-3 视图菜单(View)

| View 菜单 | | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|---|
| Workspace | | | Alt+0 | 打开工作区窗口 |
| Output | | | Alt+2 | 打开输出窗口 |
| Debug Windows | Watch | | Alt+3 | 打开 Watch 窗口 |
| | Call Stack | | Alt+7 | 打开 Call Stack 窗口 |
| | Memory | | Alt+6 | 打开 Memory 窗口 |
| | Variables | | Alt+4 | 打开 Variables 窗口 |
| | Registers | | Alt+5 | 打开 Registers 窗口 |
| | Disassembly | | Alt+8 | 打开 Disassembly 窗口 |
| Prev Error | | | Shift+F4 | 跳至前一个错误 |
| Next Error | | | F4 | 跳至下一个错误 |

调试菜单和调试命令

附表Ⅱ-4 调试菜单和调试命令（Debug）

| Debug 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| Build | | F7 | 创建一个工程 |
| Rebuild All | | | 重建整个工程忽略 independencies |
| Download | | F8 | 下载.bin 到 rom-emulate |
| Reset | | Ctrl+F5 | 重新开始程序 |
| Go | | F5 | 开始或继续程序 |
| Break | | F5 | 停止程序运行，插入调试 |
| Stop Debugging | | Shift+F5 | 停止调试程序 |
| Single | | F11 | 单步执行到下一句 |
| Step Over | | F10 | 执行单步越过函数 |
| Step Out | | Shift+F11 | 执行单步跳出当前函数 |
| Run to Cursor | | Ctrl+F10 | 运行程序到指针所在行 |
| PC to Cursor | | F12 | 运行到鼠标所指行 |
| Breakpoint | | F9 | 插入或删除断点 |
| Breakpoints | | Alt+F9 | 编辑程序中的断点 |
| Remove all Breakpoints | | Ctrl+Shift+F9 | 删除所有断点 |
| Fill RAM | | | 通过数据填充 RAM |
| Animate Single | | | 自动单步走 |
| Animate StepOver | | | 自动越过函数单步走 |
| Prov Single Trace | | Ctrl+Shift+F11 | 跟踪前一条语句 |
| Prov Trace | | Ctrl+Shift+F3 | 跟踪至前 64 条指令 |
| Next Trace | | Ctrl+F3 | 跟踪到下一条语句 |

应用菜单

附表Ⅱ-5 应用菜单与应用命令（Utility）

| Utility 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| Report | | | 将.SN8/.BIN 文件翻译为.RPT 文件 |
| Output.HEX | | | 将.SN8/.BIN 文件翻译为.HEX 文件 |
| Add Print Port… | | | 给 ICE 应用添加打印端口 |
| Easy Writer | | | 配合 Easy Writer 可烧录 IC |
| LCD Simulator | | | LCD 仿真 |

视窗菜单

附表Ⅱ-6 视窗菜单 (Windows)

| Windows 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| Cascade | | | 设置窗口层叠 |
| Tile | | | 设置窗口为非层叠 |
| Arrange Icons | | | 在窗口底部排列图标 |
| 1~9 | | | 激活选中的窗口对象 |
| Windows… | | | 当前窗口操作 |

帮助菜单

附表Ⅱ-7 帮助菜单（Help）

| Help 菜单 | 工具栏 | 快捷键 | 描述 |
|---|---|---|---|
| About Assembly… |  | | 显示程序信息、版本号及版权 |

# 附录III 伪指令表

| 伪指令 | 说明 |
|---|---|
| CHIP, ENDP | 程序开始和结束 |
| EQU, =, TEXTEQU CATSTR, SUBSTR, SIZESTR, INSTR | 变数表示法 |
| . CODE, . DATA, . CONST | 定义段 |
| ORG, . ALIGN | 设置程序和数据的地址 |
| DS | 数据定义 |
| DW | 定义一个WORD |
| INCLUDE, INCLUDEBIN, INCLUDESTD | 包含另一个文件 |
| TITLE | 用户标题定义 |
| MACRO, EXPAND, ENDM, REPEAT, FOR, FORC, EXITM | 宏 |
| . LIST, . NOLIST, . LISTIF, .NOLISTIF, . LISTMACRO, . NOLISTMACRO, . LISTMACROALL | 列表文件控制 |
| IF, IFE, IFB, IFNB, IFDEF, IFNDEF, IFIDN, IFDIF, IFIDNI, IFDIFI, ELSEIF, ELSEIFE, ELSEIFB, ELSEIFNB, ELSEIFDEF, ELSEIFNDEF, ELSE, ENDIF | 条件编译控制 |
| . ERR, . ERRE, . ERRNZ, . ERRB, . ERRNB, . ERRDEF, .ERRNDEF ERROR, ECHO | 条件编译错误显示 |
| . EXEC | 执行外部程序 |
| @BIT, @INT, @FIELD | 位运算 |

# 附录Ⅳ 图片列表

(以下可点击链接到相应图片位置)

第一章



第二章

# 附录Ⅴ 相关FAQ

请参考sonix FAQ：
http://www.sonix.com.tw/sonix/MCU_FAQ.jsp

总公司
地址：台湾新竹县竹北市台元街 36 号 10 楼之一
电话：886-03-5600 888
传真：886-03-5600 889
松翰科技（深圳）有限公司
地址：深圳市南山区高新技术产业园南区 T2-B 栋 2 楼
电话：86-755-2671 9666
传真：86-755-2671 9786
台北办事处
地址：台北市松德路 171 号 15 楼之 2
电话：886-2-2759 1980
传真：886-2-2759 8180
香港办事处
地址：香港新界沙田乡宁会路 138＃新城市中央广场第一座 7 楼 705
电话：852-2723 8086
传真：852-2723 9179
技术支持
Sn8fae@SONiX.com.tw